Learning to Control Differential Evolution Operators

Mudita Sharma

PHD THESIS

UNIVERSITY OF YORK

COMPUTER SCIENCE

October 2019

To my family, Rakesh, Maya, Manika and Deepak

Abstract

Evolutionary algorithms are widely used for optimsation by researchers in academia and industry. These algorithms have parameters, which have proven to highly determine the performance of an algorithm. For many decades, researchers have focused on determining optimal parameter values for an algorithm. Each parameter configuration has a performance value attached to it that is used to determine a good configuration for an algorithm. Parameter values depend on the problem at hand and are known to be set in two ways, by means of offline and online selection. Offline tuning assumes that the performance value of a configuration remains same during all generations in a run whereas online tuning assumes that the performance value varies from one generation to another.

This thesis presents various adaptive approaches each learning from a range of feedback received from the evolutionary algorithm. The contributions demonstrate the benefits of utilising online and offline learning together at different levels for a particular task. Offline selection has been utilised to tune the hyper-parameters of proposed adaptive methods that control the parameters of evolutionary algorithm on-the-fly. All the contributions have been presented to control the mutation strategies of the differential evolution. The first contribution demonstrates an adaptive method that is mapped as markov reward process. It aims to maximise the cumulative future reward. Next chapter unifies various adaptive methods from literature that can be utilised to replicate existing methods and test new ones. The hyper-parameters of methods in first two chapters are tuned by an offline configurator, *irace*. Last chapter proposes four methods utilising deep reinforcement learning model. To test the applicability of the adaptive approaches presented in the thesis, all methods are compared to various adaptive methods from literature, variants of differential evolution and other state-of-the-art algorithms on various single objective noiseless problems from benchmark set, BBOB.

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I would like to thank my supervisors Dimitar Kazakov and Manuel López Ibáñez for their constant guidance and motivation. I thank Dimitar Kazakov for giving me this opportunity and Manuel López Ibáñez for his reviews and insightful comments on the papers we have written together. I acknowledge their consistent support for the full period of the PhD. I thank Susan Stepney for her advice during the progression meetings and for reviewing this thesis.

Thanks to Alexandros Komninos for all the interesting discussions that resulted in important contributions. I also want to thank my friends, Taghreed Alqaisi, Chaitanya Kaul, Marcelo Sardelich and Nils Mönning, in the Artificial Intelligence group for providing the moral support and a friendly work environment in the Computer Science department.

I am grateful to the Engineering and Physical Sciences Research Council (EPSRC) for funding this project.

Finally, I want to express gratitude to my lovely parents for believing in me and supporting throughout the PhD studies. I would never have been able to reach where I am now without their support.

Mudita Sharma

Oct 2019, York, England

Authors declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

The first paper, presented in Appendix A, resulted from the explorations in the swarm based Artificial Bee Colony (ABC) algorithm. The following paper resulted from this research is published at the GECCO conference (15-19 July 2017): **Mudita Sharma and Dimitar Kazakov. "Hybridisation of artificial bee colony algorithm on four classes of real-valued optimisation functions." Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO 2017). ACM, 2017, Berlin, Germany.**

This research made it clear that parameter setting is a key issue that impacts the performance of evolutionary algorithms. It has been researched for decades but there is no general framework that can simplify the process of setting the parameters of an algorithm. The following papers resulted from the research in the direction to control a set of operators in the differential evolution generalising on a large set of problems.

Mudita Sharma, Manuel López Ibáñez and Dimitar Kazakov. "Performance Assessment of Recursive Probability Matching for Adaptive Operator Selection in Differential Evolution." 15th Intl Conf. on Parallel Problem Solving from Nature (PPSN 2018): 8-12 Sep 2018. Coimbra, Portugal.

Above paper forms the basis for the Chapter 5.

Mudita Sharma, Alexandros Komninos, Manuel López Ibáñez and Dimitar Kazakov. "Deep Reinforcement Learning Based Parameter Control in Differential Evolution." Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO 2019). ACM, 13-17 July 2019, Prague, Czech Republic.

This paper forms the basis for the Appendix B and Chapter 7. The co-author, Alexandros Komninos, contributed background knowledge of Deep Reinforcement Learning methods and performed statistical tests.

Part I

Introduction

Chapter 1

Introduction and Motivation

Nature Inspired Algorithms (NIAs) is a class of algorithms that simulate the natural processes present in nature. Evolutionary Algorithms (EAs) are a type of NIA, eg. Genetic Algorithm (GA) [140] and Evolutionary Strategy (ES) [137, 146], are inspired by the biological evolution of species. Other popular NIAs are Particle Swarm optimisation (PSO) [37] imitate the movement of bird flock or fish school, Ant Colony Optimisation (ACO) [36] is based on the idea of ant foraging by pheromone communication to form paths and Artificial Bee Colony (ABC) algorithm [88] simulate bees' foraging behaviour. These are among the efficient population based algorithms widely used for numerical optimisation. They are consistently used by researchers as a starting point for the design of a new algorithm to improve its performance. The improvement in performance is a result of introduction of their variants [10, 169, 95] or their hybridisation with other algorithms sharing similar properties [106, 110, 170]; and modifying them to implement in a specific application of an area.

Evolutionary techniques have been widely used for function optimisation by evolutionary and artificial intelligence community. EAs are derivative free algorithms that have proved to be useful for a wide range of problems in different application areas [52, 98, 79, 55, 53, 54]. They are known to guarantee near optimum solution for a given problem. The nature of the optimisation problem considered is usually black-box where the algorithm is unaware of the properties of the problem such as continuous, convex, uni modal, multi modal, separable, quadratic, high dimensional, gaussian noise, and so on. NIAs consist of two classes of algorithms: single-solution-based and population-based. As the name implies single-solution based algorithms start with only one solution e.g. Simulated Annealing [1], hill climbing [82], (1+1)-ES [146]. Population-based algorithms (PBAs) are randomised algorithms which start the search for the optimum solution using a set of candidate solutions called population such as GAs and PSO. In either case, the initial solution(s) is improved over the course of iterations according to their exploration and exploitation operators. PBAs have proved useful compared to single-solution based algorithms as the set of candidate solutions sharing information about the search space enhances their capability to find better solutions in less time. Due to their stochastic nature, it results in jumps toward the promising part of the search space which can help finding local or global optimal solutions. Crossover and mutation are two operators commonly used by these algorithms. Crossover exploits the neighbourhood of existing solutions (exploration) and mutation is responsible for exploring new regions of the search space (exploitation).

Selecting the types of crossover and mutation boils down to the problem of parameter setting. These two parameters are known as discrete parameters as they have countable number of choices, each of which have their own parameters such as a crossover rate and mutation rate respectively. These sub-parameters have continuous domain and have infinite number of choices, thus known as continuous parameters. The performance of an EA highly depends on the selection of the parameter value(s) of an EA [34, 27, 32, 114]. Considering the number of parameters involved in an EA and large number of options available for each of them, exhaustive search (manual tuning) is no longer an efficient approach. It is time-consuming and still does not guarantee near-optimal parameter values. There are two popular ways to set the parameter values, parameter tuning and parameter control. Tuning methods, also known as offline learning, are used to set parameter values before running the algorithm whereas control methods, also known as online learning, learn to decide the parameter values during the run of an algorithm. Both approaches have been explored in literature to a great extent and this is still an active area with many open questions. Tuning is a time consuming process and does not guarantee optimal parameter values. Different parameter values are shown to be optimal at different stages of an EA run [13, 42]. Thus, controlling approaches can be seen as a way to overcome the problems attached to tuning.

Within online parameter control, methods can either be rule based, feedback informed or self-adaptive. Rule based methods include a formulae that can be generation dependent. For instance, a formulae is a mathematical definition that can decide the value for mutation rate based on the current generation [51]. Feedback methods learn to make a decision on a parameter depending on the performance or feedback received from an EA [28]. Self-adaptive methods encode the parameter in the candidate solution. It then lets the parameter to evolve with the evolution of solutions as a result of the application of an operator [33].

Although parameter control is an effective approach to select parameter values on-the-fly, there are no general guidelines to follow to propose an online method. Various methods have been proposed in literature to control parameters, yet there is no widely accepted generic approaches within an EA that can be utilised for optimising functions with different properties. There are many different combinations possible from these existing methods that can be tested to boost the performance of an evolutionary algorithm. Limited combinations have been analysed and tested on a problem set but they come from few popular methods ignoring the components of old methods. The directions of research for controlling the parameters are discussed in Eiben et.al paper [101].

The context of contribution is the adaptive parameter control for parameters with discrete number of choices. The specific evolutionary algorithm studied in this thesis is Differential Evolution. It has a finite number of operators that can be selected to evolve a solution present in a population. An operator is also known as mutation strategy in DE. In the next section we briefly describe the contributions made towards the adaptive selection of an operator from a set of mutation strategies.

1.1 Thesis contributions

We present three main contributions along with two minor contributions in the field of evolutionary computation. Chapters 5, 6 and 7 describe the main contributions for the parameter control of operators in DE. The proposed methods are tested on BBOB noiseless test bed. Appendix A presents the early work done during the PhD to improve the performance of Artificial Bee Colony algorithm. Appendix B tests the applicability of an adaptive approach presented in Chapter 7 on a different CEC2005 test bed.

The first main contribution is presented in Chapter 5. It demonstrates a novel approach for discrete parameters known as Adaptive Operator Selection (AOS). Major steps in AOS method comprise credit assignment and operator selection. Credit assignment assigns a reward value to each operator and based on the performance quality, a probability is assigned to each operator. The probability of each operator is used to make a selection of an operator in the next generation. The selection mechanism is invoked for each solution in the population. *RecPM* is a proposed Operator Selector which is a variant of Probability Matching. They differ in the assignment of quality to each operator. In *RecPM*, quality assignment is inspired by Bellman equation in Markov Reward Processes, a concept in Reinforcement Learning. It aims to maximise the future reward for an operator. *RecPM* is combined with a well-known reward definition to give an AOS method named *RecPM-AOS*.

There is a lot of research on selecting discrete parameters on-the-fly [86, 85, 31, 2], but there is no method that is widely accepted to control parameters in one or many EA algorithms. These methods have a limitation to work with a particular EA and a specific test bed. Thus, in Chapter 6 we made a step towards bringing AOS methods at a common platform. We propose a unifying AOS framework of online methods for parameters with discrete choices. We simplified the working of existing AOS methods and divide each of them into a number of simplified components. Each component comprise of a number of choices. These choices mostly come from AOS methods in literature and we also propose new choices for these components. The resultant framework of different components with their choices can be utilised to combine different choices from each component to explore the AOS search space. Due to large AOS component space, it is not possible to manually tune this component space for a given set of problems. Thus, we decide to employ a racing tuner, *irace*, to explore the component space as well tune the hyper-parameters of the choices and parameters of DE. *irace* returns an appropriate AOS method for an EA and the problem set. This contribution demonstrates that offline and online setting can be utilised at different levels so as to mitigate the drawbacks of tuning and utilising the benefits of controlling the parameters online.

The design of AOS methods limits their learning ability by including the limited feedback information in the process. They usually learn from atmost three performance features of the parameters. For example, it considers either success rate in fitness improvement w.r.t median population fitness or sum of rank of raw solution fitness or quality and diversity of fitness improvement w.r.t parent fitness. In Chapter 7, we demonstrate a procedure to utilise more than three features at once to learn to decide the parameter selection. We designed 199 features that the proposed model can learn from. This huge amount of data coming at every step from an EA can be easily handled by Deep Reinforcement Learning (DRL) models. DRL utilises these features as a state representation and make an informative decision on the parameter to use, termed as action. Each action is followed by assigning a reward value to the parameter utilised at that stage. Thus, state action and reward are three key concepts utilised in DRL. The proposed method employing definitions of state, reward and action to control operators in DE is named as DE-DDQN. We explore four reward definitions and compare their performance on BBOB benchmark set.

Appendix A presents a hybrid algorithm, named as mutated Artificial Bee Colony algorithm, with the aim to improve the performance of artificial bee colony algorithm. ABC has three main phases that are responsible for maintaining exploration and exploitation of the solution search space. Two equations are derived from the existing equation present in the employed and onlooker bee phase in the standard ABC algorithm. To improve the search capability of ABC algorithm, mutation rate as in Genetic Algorithms is introduced in these search equations. The third phase, scout bee phase, is responsible for exploration and is modified to prevent losing a good candidate. Lastly, the known parameters and new proposed parameters of ABC algorithm are manually tuned.

Appendix B presents experiments of DE-DDQN on CEC2005 problem set.

1.2 Research aims of the thesis

The work presented in this thesis aims to answer the following research questions:

Q1: Can AOS be mapped to an MRP (Markov Reward Process) to maximise the future reward of operator selection?

Q2: Can the different components of AOS be generalised and generate new AOS from existing ones which improve their performance?
Q3: Can Deep Reinforcement Learning be applied to learn to decide the selection of DE operators?

1.3 Thesis Structure

The rest of the thesis is divided into three major parts. The next part includes three chapters that cover preliminary background and methods for adaptive selection of operators. In particular, Chapter 2 explains the problem of single objective optimisation, gives details on popular EA algorithms and popular methods in parameter tuning and control. Chapter 3 provides detailed discussion on the methods in parameter control for discrete parameters. This chapter presents literature for both heuristic based and Reinforcement Learning (RL) based control approaches. Chapter 4 provides the preliminary information needed for the experiments in the chapter of contributions. Thus, Part II forms a basis for understanding the baselines for comparison and the contributions presented in the thesis.

The third part (Part III) covers the three chapters of contribution in the context of online selection of mutation strategies in differential evolution. Chapter 5 describes a method based on probability matching that is inspired by a concept in RL known as Markov reward Process. The goal is to improve the performance of DE by maximising the future cumulative reward. Chapter 6 gives a unified framework for AOS methods. The framework is build by classifying and generalising its component choices. A number of AOS methods can be replicated from the framework and a large number of unique AOS designs are possible. An offline configurator is employed at top level to select an adaptive method for a given problem bed. The last chapter of contribution (Chapter 7) uses Deep Reinforcement Learning with many landscape and history features to learn the adaptive selection of DE operators. It also presents and compares four unique reward definitions with Deep RL. Each chapter of contribution presents experiments and discussion of results on BBOB noiseless functions.

We summarise the contributions in Chapter 8 and present the scope of future work.

Last Part V of the thesis, presents four appendices. Appendix A presents an improved swarm based algorithm, mutated ABC. The improvement is a result of changes made in the different phases of standard ABC algorithm. Lastly, Appendix B discusses results obtained from running DE-DDQN on CEC2005 problem set. Appendix C and Appendix D show average run time tables and operator selection with best fitness graphs for various algorithms respectively.

Part II

Background

Chapter 2

General background

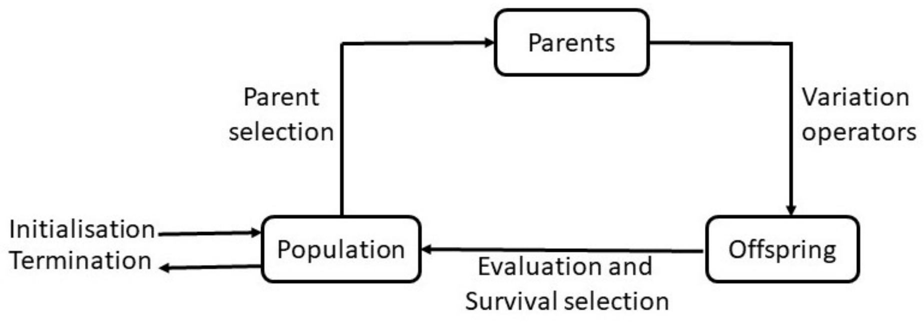
This chapter focuses on the general background on Evolutionary Algorithms (EAs), that act as basis for the experiments and the results discussed in part III of the thesis (chapters of contribution). We start by introducing the optimisation problem in the single objective scenario. It is the general problem we are trying to solve optimally. Next, we describe various popular EAs, their parameters and variants. Some of the well-known vanilla algorithms discussed are Genetic Algorithms(GAs), Differential Evolution (DE), Evolutionary Strategy (ES) and Artificial Bee Colony (ABC) algorithm.

This discussion is followed by different approaches to set the parameters in EAs. Parameter setting can be approached as a parameter tuning or controlling problem. Information flow in parameter tuning is presented followed by the classification of parameter control- deterministic, adaptive and self-adaptive. We comment on parameter tuning issues and suggest solutions to it. A few controlling methods have utilised a Reinforcement Learning architecture to adapt parameters in evolutionary algorithms. Thus, we finish the chapter with a brief introduction on Reinforcement learning architecture and its key terminologies.

2.1 Optimisation

The problem of optimisation is defined as finding the minima or maxima solution among a set of solutions in the feasible region. A feasible region is bounded by the bounds of the parameter space. Solutions are evaluated on an objective function which informs the fitness of the solution. Objective can be to minimise cost, maximise reliability, minimise effort, minimise risk, etc. Real-world problems highly

FIGURE 2.1: Evolutionary Algorithm template



depend on optimising one (single-objective) or more (multi-objective) of these objectives [171, 20, 129, 161, 83, 81].

We focus on the single objective problem which has one objective to be optimised within parameter boundaries that define the search space. Thus, it is free from constraints. This search space consists of candidate solutions $x \in X$, where each solution has a fitness or cost attached to it given by $f\colon X \rightarrow \mathbb{R}$. It is common practice to implement and test EAs on toy problems. The simplest example is optimising the sphere function, a unimodal function, with n-parameters. More complex functions can be non-linear and non-smooth functions. In case of minimisation problem, task is to find the solution with least cost x^* s.t. $f(x^*) \leq f(x), \forall x \in X$ whereas in case of maximisation, the algorithm looks for a solution with largest possible cost x^* s.t. $f(x^*) \geq f(x), \forall x \in X$.

2.2 Evolutionary Algorithms

Evolutionary algorithms (EAs) have proven to be useful as optimisation techniques [56]. They imitate the behaviour of Darwinian evolution in nature. Many popular EAs follow a common template as can be seen in Figure 2.1. The initial population, consisting of candidate solutions, is generated randomly or using some predefined heuristic. A parent selection mechanism is employed to select parents from the current population such as tournament, fitness proportional, uniform or best selection. Parents reproduce to produce a set of candidate solutions known as offspring. EAs have different ways to evolve population, usually with the help of variation operators, mutation and crossover. These operators are responsible to balance exploration and exploitation in the fitness landscape. Exploration is responsible for searching unexplored regions of the search space and exploitation searches the neighborhood of existing solutions. Some of the solutions from the parent and offspring population are chosen to to move to the next generation by a survival mechanism. Choices for survival mechanism can be generational, tournament or uniform. This process repeats while a termination criterion is not satisfied. Further in this section, we review some of the popular EAs following this template. We discuss different ways to evolve the current population and selection of the next population.

2.2.1 Genetic Algorithms

Genetic Algorithms (GAs), first proposed by Holland [140], are popular algorithms for the application of combinatorial problems. They are based on the concept of natural selection of best chromosomes which further reproduce better offspring. They imitate human biological process where each chromosome is made up of genes. Genes are significant in the process of reproduction. They represent a human characteristic for instance eye colour and gene value (called allele) can be represented by the color of eye as either green or blue. Thus, both green and blue eye colour belong to the same eye color gene. Genotype represents information on genes whereas phenotype is physical appearance. After reproduction, offspring carry genes of parents to reflect their characteristics.

In GA, chromosomes/individuals are represented as a binary string. These strings are made up of gene values each of which can be either 0 or 1. Each gene in a chromosome represents a dimension of a function to be optimised. Floating-point representation has also been investigated in literature where chromosomes are represented as a vector of real values. Floating-point representations have proved to give faster and better results than bit strings [80]. Two or more of these chromosomes are selected as parents using a parent selection mechanism. They undergo crossover and mutation to form new chromosomes. Crossover and mutation operations help maintain a good balance between exploration and exploitation and are responsible for searching different parts of fitness landscape of an optimisation problem. Based on a survival selection mechanism, individuals with better fitness survive to next generation. Both survival selection and parent selection are done on the basis of the

Algorithm 1 The working of Genetic Algorithm

```
1: Initialise parameter values of GA: $NP, S, CT, C, MT, \mu, G$
2: $g = 0$ (generation number)
3: Initialise and evaluate population P(0)
4: while stopping condition is not satisfied do
5:     Select parent population from current population P(g) biasing selection toward individuals with higher fitness
6:     Evolve parents using crossover and mutation operators
7:     Perform survival selection to form P(g+1)
8:     $g = g + 1$
```

phenotype of an individual which is based on the genotype of a chromosome. In particular, in the selection step GA performs genotype-phenotype mapping which tells how fit an individual is relative to the other individuals in the population. A phenotype represents a vector constituting n-real values in search space. The steps of GA can be seen in Algorithm 1.

The first step in genetic algorithm is important as it initialises its parameters which highly determine the performance of GA. These parameters are discussed below one by one,

- Population Size (NP)- It determines the number of chromosomes in a population which can be static or dynamic throughout the run. Although large population size discourages premature convergence, it is computationally expensive due to the large number of function evaluations per generation. On the other hand, GA does not perform well with small population size [61] because it prevents diversity in the population. Adaptive population size has been suggested between 10 to 160 with increments of 10 [61].

- Selection mechanism (S)- Parent selection is performed to select parents for evolution and survival selection for the formation of next generation among parents and offspring. For selection of a parent, binary tournament randomly samples two parents from the current population. The one with better fitness survives to become a parent. To form the next generation, individuals based on fitness ranking can be selected. Other techniques are random and fitness-proportionate selection.

- Type of crossover (CT)- Crossover operator (also known as recombination operator) acts upon selected chromosomes to pass its genes to produce better offspring than parents. This can be done in many ways. One-point and two-point crossover cut the parents at randomly chosen one and two points respectively and exchange them to form offspring. In uniform crossover, genes are randomly exchanged between two parents. Good parts in the parents can be lost with high number of crossover points, thus two-point crossover is usually the first choice.

- Crossover Rate (C)- It is the rate with which crossover operator is applied on two or more chromosomes. In each generation, $C \cdot NP$ chromosome undergo crossover. With high value of C, good performing genes in a chromosome can be lost while a low value can lead to stagnation in population due to low exploration. Widely accepted crossover rate varies from 0.25 to 1.0 with increments of 0.05 [61] depending on the problem.

- Type of mutation (MT)- Mutation flips one or more bits in the offspring. In one-point mutation, one bit is randomly flipped with mutation probability μ. Inversion is another mutation operator where two random bit positions are selected and all bits between these positions are reversed. It is responsible for maintaining genetic diversity in a generation.

- Mutation rate (μ)- It can be applied at two levels- allele and chromosome level. At allele level, $\mu \cdot NP \cdot CL$ alleles undergo mutation where CL is the chromosome length. In the case of chromosomes in a generation, any random bit in each $\mu \cdot NP$ chromosomes undergo bit-flip.

- Generation gap (G)- It decides the number of individuals to be replaced from the current population. $G = 0.5$ indicates that half of the individuals in the current population will be replaced by offspring. When G is set to 1.0, all the candidates in the current population will be replaced by the offspring to form new parent population.

2.2.2 Evolution Strategies

Traditional evolutionary strategy (ES) [137, 146] generates one offspring from one parent, known as (1+1)-ES. A solution in ESs is represented as a vector of real values. The offspring is generated using mutation which induces some noise to the parent, x_i, using Gaussian process with mean 0 and predefined standard deviation (or mutation step size).

$$x_i = x_i + N(0, \sigma_i) \tag{2.1}$$

Each dimension value of the solution is evolved with the help of independent Gaussian distributions. The mutation step size is adapted using $1/5$-success rule [139], which says, the ratio of successful mutations to all mutations should be $1/5$. For greater ratios increase the step-size, else decrease it. This is a simple example of adaptation of mutation rate in $(1+1)$-ES.

$(\mu + \lambda)$-ES was proposed by Rechenberg in 1973 [138] where instead of one parent and offspring, it utilises a population of parents (μ) to evolve a population of offspring (λ). The best solutions among parent and offspring population survive to become next generation parent population of fix size μ. (μ, λ)-ES [145] is another variant of evolution strategy where parent population in next generation is formed by selecting candidates from λ offspring.

One of the well-known ES variant is known as Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [64]. It is known to solve difficult non-linear, non-convex black-box optimisation problems in continuous domain [63, 65]. It has invariance properties to preserve order transformations of the objective function value and to preserve angle transformations of the search space including rotation, reflection, and translation. These properties ensure uniform behavior on classes of functions and generalize on the empirical results. CMA-ES is extended to BIPOP-CMA-ES [62]. After a first single run with default population size, it applies two interlaced multistart regimes, each equipped with a function evaluation budget accounting for the so far conducted function evaluations. Depending on which budget value is smaller, a complete run of either one or the other strategy is launched [62]. Another promising variant of ES is Natural Evolutionary Strategy (NES) [144]. It provides an adaption mechanism to update the learning rate for step-size of distribution, by ascending the gradient towards higher expected fitness.

2.2.3 Differential Evolution

Differential Evolution (DE) [132], like GA, is a population-based algorithm. As in ES, the population in DE consists of real-valued solutions in the search space. DE utilises vector differences for evolving solutions known as mutation strategy. It follows a nomenclature to differentiate one *DE* from others in their definition of mutation strategy and crossover operator. The nomenclature scheme is followed to reference the different *DE* variants that is of the form “DE/x/y/z”. For instance in “DE/rand/1/bin”, “DE” means Differential Evolution, “rand” refers to the random individuals selected to compute the mutation values, “1” is the number of pairs of solutions chosen and lastly “bin” indicates a binomial crossover of selected individuals. Another choice for z is “exp”, an exponential crossover.

The procedure of DE is shown in algorithm 2. It starts with a population of candidate solutions, x_i generated randomly in the search space of dimension n. Thus, each candidate in the population is represented as n-dimensional vector. DE does not employ parent selection mechanism, that is each parent mutates with two or more solutions from the population using a mutation strategy to create an offspring solution u_i (or often known as trial vector). A mutation strategy is a linear combination of three or more parent solutions x_i, where i is the index of a solution in the current population. Some mutation strategies are good at exploration and others at exploitation, and it is well-known that no single strategy performs best for all problems and for all stages of a single run [50]. Many mutation strategies have been proposed to generate offspring which differ in the solutions used as a linear combination of each other. These following mutation strategies [132] are frequently used in the literature for optimisation problems. They have their own way to balance exploration and exploitation in the search space:

"rand/2":	$u_i = x_{r_1} + F \cdot (x_{r_2} - x_{r_3} + x_{r_4} - x_{r_5})$
"best/1":	$u_i = x_{best} + F \cdot (x_{r_1} - x_{r_2})$
"curr-to-best/1":	$u_i = x_i + F \cdot (x_{best} - x_i + x_{r_1} - x_{r_2})$
"best/2":	$u_i = x_{best} + F \cdot (x_{r_1} - x_{r_2} + x_{r_3} - x_{r_4})$
"rand/1":	$u_i = x_{r_1} + F \cdot (x_{r_2} - x_{r_3})$
"rand-to-best/2":	$u_i = x_{r_1} + F \cdot (x_{\text{best}} - x_{r_1} + x_{r_2} - x_{r_3} + x_{r_4} - x_{r_5})$
"curr-to-rand/1":	$u_i = x_i + F \cdot (x_{r_1} - x_i + x_{r_2} - x_{r_3})$
"curr-to-pbest/1":	$u_i = x_i + F \cdot (x_{pbest} - x_i + x_{r_1} - x_{r_2})$
"curr-to-pbest/1(archived)":	$u_i = x_i + F \cdot (x_{pbest} - x_i + x_{r_1} - x_{archive})$

where r_1, r_2, r_3, r_4, and r_5 are randomly generated mutually exclusive indexes within $[1, NP]$, and NP is the population size. u_i and x_i are the i-th offspring and parent solution vectors in the population respectively, x_{best} is the best parent in the population. An additional numerical parameter, the crossover rate ($CR \in [0, 1]$), determines whether the mutation strategy is applied to each dimension of x_i to generate u_i. At least one dimension of each x_i vector is mutated (represented as j_{rand}) in Algorithm 2. F, known as scaling factor is the mutation rate that acts as a weight to the set of pair of solutions selected to calculate the mutation value. "rand/1" and "rand/2" combine one random vector with one and two random weighted difference vectors respectively. They are good at exploration of the search space. "best/1" and "best/2" explore the neighborhood of the best candidate. "curr-to-pbest/1" and "curr-to-pbest/1(archived)" [169] are modification of "curr-to-best/1". "curr-to-rand/1" and "rand-to-best/2" explore the neighborhood of current parent and best parent in the current population respectively.

After the parent is evolved using a mutation strategy, the fitness of the offspring is compared with its parent and a less fit solution among parent and offspring is replaced by the more fit candidate. Thus, DE employs greedy selection mechanism.

Many algorithms are used for numerical benchmark optimization, but DE has shown better performance than genetic algorithms and particle swarm optimization for continuous problems, for instance see papers [156, 148]. A survey of development in DE can be found in [30] and for applications of DE refer to [6, 108, 23, 29].

Algorithm 2 Differential Evolution algorithm

```
 1: Initialise parameter values of DE: $NP, CR, F$
 2: Randomly initialise and evaluate the population
 3: Generation $g = 0$
 4: **while** stopping condition is not satisfied **do**
 5:    **for** each candidate in population **do**
 6:       Uniformly select $r_1 \neq r_2 \neq r_3 \neq r_4 \neq r_5 \neq i$
 7:       Generate a mutated vector, $u_{i,j}$
 8:       $j_{rand} = random[1, n]$
 9:       **if** $random[0, 1) < CR$ *or* $j == jrand$ **then**
10:          $v_{i,j} = u_{i,j}$
11:       **else**
12:          $u_{i,j} = x_{i,j}$
13:    Evaluate the population $u_{i,j}$
14:    Greedy selection between $x_i$ and $u_i$
15:    $g = g + 1$
```

2.2.4 Artificial Bee Colony Algorithm

Artificial Bee Colony (ABC) algorithm [88] is a population based algorithm, proposed by Dervis Karaboga in 2005. It imitates the foraging behaviour of bees in the environment. In ABC, the position of a food source represents a solution in the search space and the nectar amount at a food source is the fitness of the solution. There are three kinds of bees considered in ABC algorithm, namely employed bees, onlooker bees and scout bees. Half of the population consists of employed bees and rest half represents onlooker bees. The number of the employed bees is equal to the number of solutions in the population.

Algorithm 3 shows the working of ABC algorithm. At the initial generation, all bees (employed and onlooker bees) in a generation are randomly distributed between the lower and the upper bound dimensional limits of the function. In the employed bee phase, each employed bee, x_i is assigned a random position, v_i in the neighbourhood of its current solution. v_i is generated by selecting and changing a random dimension of x_i as shown below. Other dimension values remain same in both positions, x_i and v_i. This phase is responsible for increasing the population diversity.

$$v_i = x_i + rand[-1, 1] \cdot (x_i - x_k), k \in [1, NP], i \neq k \tag{2.2}$$

where x_k represents the randomly selected k-th employed bee and rand$[-1, 1]$ is a random number in the range -1 and 1. Each employed bee uses greedy selection to select a position with better fitness value among x_i and v_i. Once the employed bee decides its position, it dances on the dance area. Dance is a medium to share the location and amount of honey present on that location with onlooker bees. Each onlooker bee observe the dance of employed bees and exploit the neighbourhood of selected employed bee position. In the onlooker bee phase, the selection of employed bee is made probabilistically (roulette-wheel selection) using the following formulae,

$$P_i = \frac{fit_i}{\sum_{j=1}^{NP} fit_j}, i \in [1, NP] \tag{2.3}$$

where P_i and fit_i denotes the selection probability and fitness of i-th employed bee in the population respectively. The better the fitness of i-th solution the higher the chance of getting selected by an onlooker bee. Each onlooker bee is assigned a random position in the neighbourhood of the selected employed bee. This phase is responsible for exploiting the existing solutions in the search space. It then greedily selects a position among the current position and the position selected in the neighbourhood of employed bee. At the end of each generation, an employed bee becomes scout bee if its position has not been changed for a predefined number of generations (known as limit). The scout bee abandons its position and is assigned a random position in the search space. There can be atmost one scout bee in a generation.

ABC has used intelligent behaviour inspired by a swarm of bees to successfully solve a range of mathematical problems [147]. ABC has been compared with DE [93] and GA [18, 89] on a number of numerical problems. ABC has also been used for constrained optimization problems [92] and has been applied for training neural networks [90, 91]. It has been utilised for industrial problems, for designing IIR filters [97] and for the leaf-constrained minimum spanning tree problem [152]. A survey of advances in ABC and its application can be found in [96].

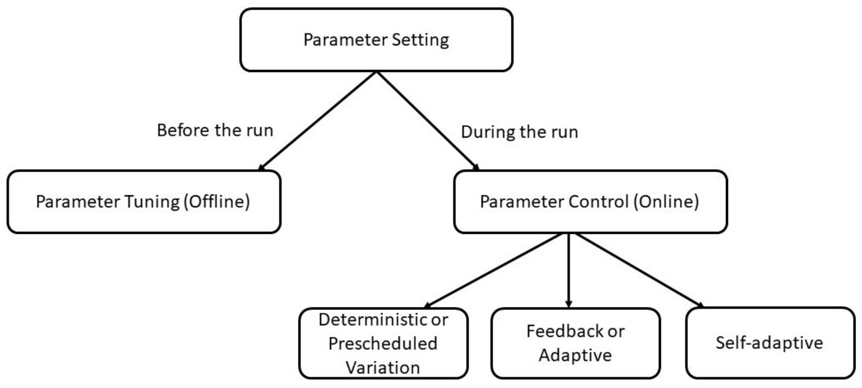
Algorithm 3 Artificial Bee Colony algorithm

```
1: Initialise parameter values of ABC: $NP$, $limit$
2: Randomly initialise the position of bees
3: Evaluate the fitness of each individual in the population
4: $g = 0$ (generation number)
5: while stopping condition is not satisfied do
6:     Employed bee phase
7:     for each $\vec{x}_i$, $i = 1, \ldots, NP/2$ do
8:         Generate a position using equation 2.2
9:         Greedy selection of a position
10:    Onlooker bee phase
11:    for each $\vec{x}_i$, $i = NP/2, \ldots, NP$ do
12:        Roulette-wheel selection of an employed bee using equation 2.3
13:        Generate a position using equation 2.2
14:        Greedy selection of a position
15:    Scout bee phase
16:    Scout bee assigned a random position in the search space
17:    $g = g + 1$
```

2.3 Parameter Setting

Performance of evolutionary algorithms highly depend on the selection of parameter values. It is challenging to find the best parameter setting of the EA in advance due to limited knowledge about the effects of parameters on EA performance. Also, an EA usually has more than one parameter and their collective behavior impacts the performance of the algorithm. An EA design involves setting parameters which can be of two types. The first type takes countably finite set of values, known as discrete parameters. Population size is one such parameter that is common to every population-based EA such as GA and DE. It has finite possibilities to select from when its range is bounded. The second type of parameter is the one that takes numerical values with infinite choices known as continuous parameters. For example crossover rate in GA whose domain is real-valued in the range $[0.0, 1.0]$. The parameters space of an algorithm can be written as a combination of all parameter choices (that is a Cartesian product). Let there be n number of discrete parameters a_1, a_2, ..., a_n and m number of continuous parameters b_1, b_2,..., b_m belonging to an EA. Thus, there are $a_1 \times a_2 \cdots \times a_n \times b_1 \times \cdots \times b_m$ number of parameter choices for an algorithm. Parameters can form a hierarchy for example the type of crossover in GA (a discrete parameter) can have choices such as single point or multipoint crossover, both of which have a continuous parameter, crossover rate, which needs to be set.

FIGURE 2.2: Parameter setting classification

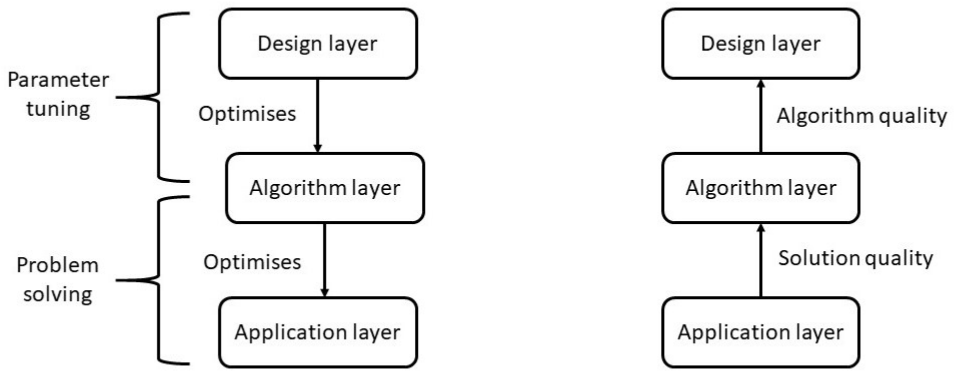

Parameter Setting
Before the run
During the run
Parameter Tuning (Offline)
Parameter Control (Online)
Deterministic or Prescheduled Variation
Feedback or Adaptive
Self-adaptive

Thus, selecting a parameter may involve deciding another parameter value attached to them.

Determining an optimum value for each parameter in a given EA can be a tedious task as the parameter domain is generally huge. Near optimal parameter values can be set manually by hit-and-trial method. This method requires a lot of computational effort and time to come up with a set of near optimum parameter values as parameters are set before the algorithm is run and it always leaves some possibility to find a better value. Thus, manual tuning of the combinations of parameters involved in an algorithm makes it practically impossible to decide the parameter values. Thus, it becomes important to employ a parameter selection method that selects the near-optimal parameter values of the algorithm.

These parameters can either be tuned using offline configurators by refining optimal configurations after a few runs (Sequential Parameter Optimisation [17]); by discarding the configurations if they are worst than others when there is enough statistical evidence to reject them (Iterated Racing [115]) or can be controlled using deterministic, self-adaptation or adaptation methods. In case of parameter tuning, an offline approach, the parameters are tuned and trained on a problem set. Once parameters are tuned the values do not change during the run of the algorithm on the test set. On the contrary, in the parameter control, an online approach, there is no training involved. The parameter values are adapted and learned on-the-fly. Thus, setting the parameters can be classified into tuning and controlling methods as suggested in [40]. The classification is shown in Figure 2.2.

FIGURE 2.3: Control and Information flow in offline tuning



2.3.1 Parameter Tuning

Parameter tuning methods have two phases, training and testing phase. These two phases have different set of problems for training and testing. In the training phase, different parameters are trained on the training set. The parameter configuration returned by tuner is used in the testing phase on the test bed. This configuration remains fixed during whole run of the algorithm during testing. A separate budget is allocated to training and testing phase. Figure 2.3 shows the three layered structure for offline setting. This structure includes design level, algorithm level and application level described in [41]. The design layer selects the appropriate parameter values for the algorithm present at the algorithm level according to the problem at hand. This can be done with the help of a tuner or using an appropriate optimiser. The algorithm layer includes an evolutionary algorithm that is chosen to optimise the problem instance at the application layer. This information can flow hierarchically in either direction referred to as control and information flow. The problem solving part tries to find an optimal solution for a problem that is, it iteratively generates candidate solutions from the problem space to find the optima of the problem. The parameter tuning part tries to find optimal parameter values for the algorithm from the parameter space.

A number of tuners following the above layered procedure are proposed in the literature. One of the well-known class of tuners inspired by [116] are known as racing methods. This class of tuners maintain a pool of configurations by discarding and retaining elite configurations using statistical tests to guide the search direction in the parameter search space. F-Race [19], Interactive F-RACE (I/F-Race) [14] and *irace* [115] are few tuners that fall under this category. Another method to tune parameters is to use an optimisation algorithm such as GA with the aim to find the best combination of parameter values given the problem in-hand [61, 8, 16]. Others include experimental design technique [27]; Local Search methods- FocusedILS [76] and ParamILS [76]; Sequential Parameter Optimisation [17] and SMAC [75]. An interested reader is referred to [41] for detail on tuning methods. A survey of manual and automated approaches can be found in [32, 43].

Despite having many tuners available to optimise the parameter space of an algorithm, these tuners suffer from some issues. Parameter tuning methods are time-consuming and need a large budget. Additionally, due to the large number of parameters and their dependence on each other it becomes difficult for parameter tuners to explore large parameter spaces. Another problem with tuning is that static parameter values given by tuner lead to sub-optimal parameter values. This is so because the parameter search space needs to be explored during the initial runs of an algorithm before focusing on certain areas of search space. This can be overcome by employing different parameters at different stages of algorithm run [153, 13, 35]. In addition to these issues, tuners generally perform poorly in black-box setting where the class of problem is not known in advance. Due to differences in complexity of landscape of problem, single parameter configuration is not suitable for all kinds of problems.

To overcome the above problems, parameter control can be an alternative approach to find the optimal configuration for different problems learned during the run of the algorithm.

2.3.2 Parameter Control

The procedure of deciding parameter values during the run of an algorithm is termed as parameter or online control. Control methods are important because parameter values perform differently at different stages of the run of the algorithm and parameter values need to change according to the fitness landscape. It has shown to overcome the drawbacks of parameter tuning by dynamically selecting the parameters for the next generation [13, 31, 70, 49].

Online selection of parameter values can be performed in three different ways [40], shown in Figure 2.2. Deterministic or pre-scheduled methods work by assigning parameter values according to predefined rules. They are uninformed methods as they do not receive any feedback from the process of running an algorithm. Thus, these methods do not generalise well on different classes of problems. One of the early algorithms for deterministically setting parameter values is Simulated annealing that adjusts the temperature based on a deterministic rule. An alternative method to above approach is self-adaptive method. It is an efficient approach to adapt continuous parameters [32]. These methods encode parameter values in the genome along the solutions and allow them to evolve with the problem solutions [153, 12] with the help of EA operators such as crossover and mutation. The encoded parameters in the form of genome are converted to parameter values and good performing parameter values lead to the production of good individuals. These individuals survive and lead to the production of better individuals with the help of better parameter values. A well-known variant of ESs, known as CMA-ES [64], uses self-adaptation of mutation step size. For brief discussion on CMA-ES see section 2.2.2. Other self-adaptive methods can be found in [107]. The class of methods that learn from the feedback received during the run of the algorithm are known as feedback or adaptive methods [69, 163]. These methods make informative decisions that give direction to the search of parameter values. This is done by capturing the progress of the algorithm from the application of their current and past performance. It then assigns a value to the parameter according to the feedback provided. *1/5 success rule* in Evolutionary Strategy is one of the first examples of control method. For details on this rule refer to section 2.2.2. A survey on adaptive methods can be found in [4]. The literature on adaptive methods for discrete parameter space is discussed in detail in the following chapter.

[168] distinguishes the performance of online control of parameters at two levels, population level or individual level. For the population level parameter control, all individuals in a generation share the same parameter value. The parameter values may vary during the evolutionary process, making it more suitable for the current evolutionary state of the population. A fuzzy adaptive differential evolution algorithm (FADE) [111] is a fuzzy model to adapt continuous parameters, F and CR, at

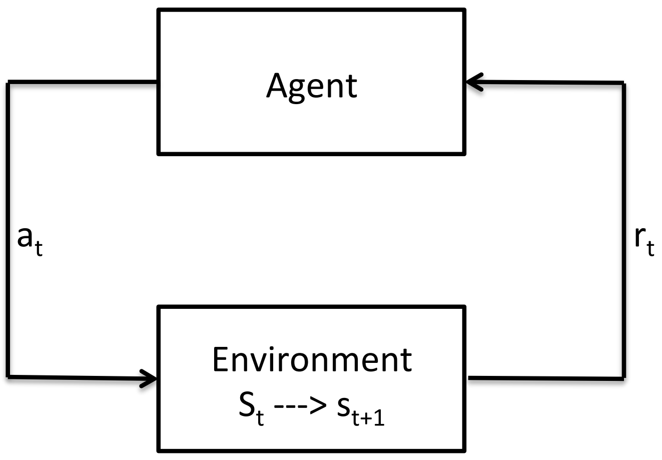
population level in DE. For individual level parameter control strategy, each individual has its own parameters, and thus the parameter values of the entire population present some distribution characteristics in the parameter space. An adaptive DE algorithm, JADE [169] uses the results of the last generation to count the mean of the parameters, F and CR, of all good individuals by the Lehmer mean, which is used to guide the distribution of parameter values for the next generation. The values of the individual parameters are assigned according to normal distribution for CR and Cauchy distribution for F based on the mean. Success History based DE algorithm, SHADE [159] is an improvement upon the JADE strategy by introducing a weighted method for the mean formula and utilising a history feature for storing the successful mean of several recent generations. SAGA [71] is another algorithm that uses both population and individual-level self-adaptive control method for population size and mutation strength in GA.

Online methods in literature are designed to focus either on specific parameter, multiple parameters (ensembles) or independent of any parameter [40]. Parameter specific control methods are formulated for a particular parameter. For instance a review of different methods proposed for population size in GAs can be seen in [114]. Control ensembles is another technique that combines different methods to create an overall method to control multiple parameters [71, 12, 121]. Parameter dependent consists of methods to control any parameter and are not specific to an parameter [167, 5]. A detailed literature for these three types of control methods can be seen in [101].

2.4 Reinforcement Learning

Reinforcement Learning (RL) is an area of machine learning to makes a sequence of decisions. It considers an artificial agent that interacts with its environment and learns with the experience to optimise the objective given in the form of cumulative rewards or panelties. While interacting with the environment it collects and records some information that is represented by a state. RL estimates the value of an action given a state called *Q-value* as the expected long-term return with discount to learn the *policy*. Thus, the ultimate goal is to learn the policy that predicts what action is to

FIGURE 2.4: Reinforcement learning architecture



be taken given the state. The general RL problem is depicted in Figure 2.4. The agent starts at a time step 0. It starts with a given state, $s_0 \in S$ within its environment. At each time step t, the agent takes an action $a_t \in A$, obtaining a reward $r_t \in R$ and moving to a new state $s_{t+1} \in S$.

Chapter 3

Adaptive Selection for Discrete Parameters

In the previous chapter, we presented a general background on various evolutionary algorithms and different ways to set their parameters. Choosing the right parameter of an algorithm is often a key for improving the performance of the algorithm. Parameters can be set using tuning or control methods. Adaptive methods are the most widely used approach to set the parameter values under parameter control. Thus, this chapter presents the literature on adaptive parameter control approach for discrete parameter space.

Discrete parameter consist of countable number of choices. These choices can be finite number of real or integer values or operators (Op). Adaptive Operator Selection (AOS) [59, 49] is the widely accepted term for the methods that select discrete parameters on-the-fly. It dynamically selects, at each generation $g + 1$ of an algorithm, an operator op at run-time from a finite set of operator choices [101]. Mathematically, let there be a set of alternative operators Op where an operator $op \in Op$ is a function $op\colon X \to X$ that takes a number of solutions (parents) and returns one or more solutions (offspring). X represents the set of all possible solutions in the population. One such operator could be the mutation operator (aka mutation strategy) in Differential Evolution (DE). It is an operator that takes two or more solutions from the parent population to generate an offspring.

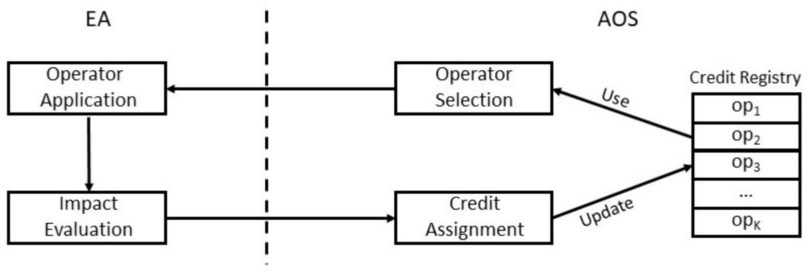
Researchers have approached adaption of parameters from two perspectives. The first approach uses sequence of heuristics collecting information from the landscape during the run of an algorithm and learn to predict the parameter values.

The selection is based on (*1*) a credit or reward value that rewards recent performance improvements attributed to the application of the operator and (*2*) an estimated quality of the operator that accumulates historical performance or takes into account the performance of other operators. The Second approach is to use a parameter independent method that is Reinforcement Learning (RL). RL is a powerful optimisation technique that imitate human learning behaviour through experience. In RL, an agent takes an action in the environment that returns the reward and the next state. The goal is to maximize the future reward which is received when an action (or make a decision) in an environment (EA) is taken. The reward is proportional to 'goodness' of the action. Once an action is taken, agent reaches a new state. This process of state action reward continues until convergence. We will review adaptive methods falling in these two categories one by one.

3.1 Adaptive Operator Selection

AOS has been studied for many years and a lot of research has been done in the context of adaptive methods to control parameter with discrete set of choices in EAs. The history of AOS can be traced back to 1990s [31]. Since then, there have been various AOS methods proposed in literature that vary broadly in various aspects such as amount of information to use from the past performance of the algorithm and whether including previous quality in the quality definition is an effective approach. Figure 3.1 shows the working of AOS [117]. The candidates in the population evolve to produce offspring with the operator application. This can be at the population or individual level. The impact of each operator is calculated based on their performance. The two key components in AOS is the Credit Assignment (CA) and the Operator Selection (OS) techniques. CA utilises the feedback provided by the EA to calculate reward/credit of operators. Credit registry (CR) can store rewards obtained in current and previous generations which are used by OS technique to make a decision on the next operator to be used by the solutions in the population.

FIGURE 3.1: Adaptive Operator Selection



3.1.1 Credit Assignment

CA defines the performance statistics that measures the impact of the application of an operator and assigns a reward value according to the impact. Impact can be based on direct offspring fitness[113, 78, 143, 99, 141, 21, 100], fitness improvement w.r.t parent [78, 72, 77, 50, 47, 131, 26, 113, 15, 120, 117, 164, 49, 118, 3], fitness improvement w.r.t current best parent [78, 77, 31], fitness improvement w.r.t median fitness [86, 85, 84] or relative fitness improvement [59, 127]. The reward of an operator is defined in terms of any of these impact from one or more generation obtained by the operator application. These impacts have been considered to combine in different ways. CA technique assigns reward to each operator after a population has evolved. [77, 165, 72, 78, 59, 47, 78] uses raw fitness as result of the impact calculation.

3.1.2 Operator Selection

OS estimates the quality of each operator based on the reward assigned to it at previous generations (stored in the CR). In the end, a selection technique is employed to select an operator for evolving an parent based on the probability assigned to each operator. The same selection technique is used to evolve all parents in a generation. As we progress with these steps, we learn more and more about the landscape and after a number of generations, this process moves the solutions in a particular search direction. There are many choices for OS techniques.

ADOPP [86] is among the initial methods proposed to control two operators (crossover and mutation) in GA. It considers the number of fitness improvements in a population over median population fitness and assigns each operator a probability proportional to the contribution made by them. [117] made an attempt to test twelve different combinations resulting from different CA and OS techniques taken from popular AOS methods. It presents AOS coupled with Adaptive Operator Management (AOM). AOM introduces a concept where operators are unborn, alive or dead based on their performance criteria. It decides whether an operator is important for current stage or not. The choices are limited to type of window and hyper-parameters of AOS methods. In the period 2010-2019, there have been a lot of interest in parameter control methods, especially adaptive methods. A number of adaptive methods for discrete number of choices are proposed in [45]. The hyper-parameters involved in these methods are tuned using an offline configurator known as *F-Race*. It also gives a high level view of an adaptive method divided into two steps as credit assignment and operator selection. [2] further classifies Adaptive methods into four categories: Feedback Collection, Parameter Effect Assessment, Parameter Quality Attribution and Parameter Value Selection. It shows methods to perform each of these four steps in multi-objective scenario.

PM-AdapSS [59], *F-AUC-MAB* [49] and *Compass* [118] are among the popular AOS methods in literature. *PM-AdapSS* considers the immediate impact/performance of the operators in the form of relative fitness improvement. It then calculates the reward, $r_{g+1,op}$, of selected operator op at generation $g+1$ as:

$$r_{g+1,op} = \frac{1}{n^{g}_{succ,op}} \sum_{i=1}^{n^{g}_{succ,op}} \frac{f_{bsf} \cdot |f(x_i) - f(u_{i,op})|}{f(u_{i,op})} \tag{3.1}$$

where $n^{g}_{succ,op}$ is the number of offspring that improved over its parent at generation g with the application of operator *op*, and $f(u_{i,op})$, $f(x_i)$ and f_{bsf} are the fitness of an offspring solution generated by selected operator op, of its parent solution and of the best solution found so far, respectively. If there is no improvement or the operator is not selected at generation g, the reward is assigned zero. It then uses probability matching (PM) to map the quality of each operator to a probability value and applies roulette-wheel selection to probabilistically choose the next operator. In particular, the quality of each operator is calculated as:

$$q_{g+1,op} = q_{g,op} + \alpha \cdot \left(r_{g+1,op} - q_{g,op}\right), \forall op \in Op \tag{3.2}$$

where α is a parameter called adaptation rate. The selection probabilities for choosing an operator in generation $g+1$ are calculated as:

$$p_{g+1,op} = p_{\min} + (1 - K \cdot p_{\min}) \left(\frac{q_{g+1,op}}{\sum_{j=1}^{K} q_{g+1,j}} \right) \tag{3.3}$$

where $p_{\min}$ is a minimum probability of selection.

Due to unknown bounds of fitness function, a comparison based (rank based) assignment is proposed in [49]. It credits the operators with the sum of the ranks of the impacts with its applications. The CA of *F-AUC-MAB* [49] uses a sliding window of size W to store the rank-transformed fitness obtained by the last W selected operators that generated an improved solution. A decay factor is applied to the ranks so that top-ranks are rewarded more strongly. The ranks in the window are used to compute a curve of the contribution of each operator and the Area Under the Curve (AUC) is taken as the reward value of the operator. The reward value of each operator are stored in the CR. Instead of using ranks, [86, 84, 135, 126, 25, 134, 150] combines the number of successful and optionally unsuccessful applications of operator in certain number of generations. The OS in *F-AUC-MAB* uses a multi-arm bandit (MAB) technique called Upper Confidence Bound (UCB) [9] to calculate quality of operator op, $q_{g+1,op}$:

$$q_{g+1,op} = r_{g+1,op} + C \cdot \sqrt{\frac{2 \log \sum_{j=1}^{K} n_j}{n_{op}}} \tag{3.4}$$

where C is a scaling factor parameter, n_{op} is the number of applications of operator op in the last W applications that improved a solution. In the above equation, $r_{g+1,op}$ introduces exploitation whereas the second term introduces exploration. The operator selector greedily chooses the operator with the highest quality value.

On the other hand, *Compass* [118] takes a different approach by considering three measures based on each operator. It calculates the quality and diversity of fitness improvement of solutions generated in last fix number of applications. The quality and diversity coordinate is projected over a line defined by an angle, θ to give reward to the operator. The final reward assigned to each operator is divided by operator's execution time and is stored in the CR. In particular, the reward of each operator is normalised after adding some noise to each operators' reward to obtain probability. Roulette-wheel selection is used to select next operator for candidates in next generation. Other OS techniques include weighted normalised sum [77, 78] and weighted sum of current and previous reward [113, 165].

There are multiple AOS methods proposed in the literature based on *reinforcement learning (RL)* techniques such as probability matching [49, 150], multi-arm bandits [59]. It has been shown that RL based AOS methods perform better than static values [102]. AOS has been approached in literature as RL which involves state, reward and action. RL comes under the category of parameter independent methods (Sec. 2.3.2) that estimates a map from states to actions (called policy), where actions are optimal in the sense that they accumulate maximum reward in a horizon. The horizon denotes the number of steps taken by an algorithm, typically infinite. Thus, RL uses feedback from the EA that describes the state of the search and implement actions as changes to parameter values. $Q(\lambda)$ learning [131], SARSA [22, 39, 143] and others [99, 141] are few of them.

AOS methods based on RL use one or more features to capture the state of the algorithm search at each generation, select an operator to be applied and calculate a reward from this application. Typical state features are fitness standard deviation and mean fitness of population, fitness improvement from parent to offspring, best fitness seen so far [39, 99]. Typical reward functions measure reward as the difference between the best fitness of the parent, f^{g}_{Best}, and the offspring, f^{g+1}_{Best} [131, 124, 21]. This can also be seen as difference between best solutions in two consecutive generations. Some researches include best offspring [141] in the denominator, whereas others consider parent fitness with the function evaluation difference [99], $Evals^{g+1} - Evals^{g}$, or computational time in the denominator [143]. It can be seen in the following equation where C represents a scaling constant,

$$C * \frac{\frac{f^{g+1}_{Best}}{f^{g}_{Best}} - 1}{Evals^{g+1} - Evals^{g}} \tag{3.5}$$

Other reward definitions have been explored in literature. Muller et al. in their paper [124] used temporal difference learning to control the mutation step size for (1+1)-ES using the 1/5 rule to define state. They tested four reward types, (i) 1, 0 or

-1 if the success rate increased, remained the same or decreased respectively, (ii) the difference of the current fitness minus the fitness of the previous step, (iii) 1, 0 or -1 if the fitness improved, remained the same or deteriorated respectively and (iv) the realised step length in the search space multiplied by the reward as defined in (iii). Definition (iv) is concluded to be best. [100] explores four reward definition with different AOS based RL methods proposed in literature. (i) same as equation 6.18 with offspring and parent fitness interchanged, (ii) 1 and 0 if the improvement is made or not resp., (iii) weighted improvement of current best fitness, and (iv) raw current best fitness. They reported (i) and (ii) as outperforming others.

Parameter control methods using an offline training phase has also been considered by researchers to collect more data about the algorithm than what is available within a single run. For example, Kee, Airey, and Cyre [104] use two types of learning: table-based and rule-based. The learning is performed during an offline training phase that is followed by an online execution phase where the learned tables or rules are used for choosing parameter values. More recently, Karafotias, Smit, and Eiben [103] train offline a feed-forward neural network with no hidden layers to control the numerical parameter values of an evolution strategy.

Chapter 4

Experimental setting

The previous chapters summarise the important algorithms in the evolutionary computation. We have also seen the specific techniques for optimising the parameters by means of tuning and controlling approaches. This chapter examines the experimental settings for evaluating the contributions.

The work presented in the thesis aims to improve the performance of Differential Evolution (DE) algorithm. Mutation strategy in DE is responsible to generate new candidate solutions and thus evolve the parent population. It has been shown that the use of single strategy in all the generations is not enough to prevent premature convergence [118]. The selection of different operators to evolve different parents have shown to perform better than utilising single operator in a run. A maximum of four operators aka mutation strategies are adapted by various methods proposed in literature to improve the search capability of DE. The adaptive algorithms used to learn to adapt four strategies, rand/1, rand-to-best/2, rand/2 and current-to-rand/1, on-the-fly can be found in [59, 135, 48]. [134, 45, 60] have also made an attempt to adapt different set of mutation strategies in DE.

We show results of the proposed AOS methods adapting the above mentioned four strategies. We further add five more operators to the list of the above mentioned operators to show the improvement of employing nine over four operators in DE. Thus, we intend to maintain population diversity and exploitation capability. We intend to increase the robustness of DE algorithm on five different function classes by optimising the selection of nine different operator choices. These operators are commonly used in literature and have shown to perform good within DE. Their mathematical formulation is shown in section 2.2.3.

In the further chapters of contribution we present three control methods. All contributions consider individual level selection of operators rather than population level. That means, operator selection mechanism is invoked for each parent solution in a generation. More details on individual, component and population level adaption can be found in [7], also discussed in section 3.1.

4.1 Problem set

We consider the single objective numerical problems w.l.o.g are to be minimised. Most of the methods try to achieve good performance on a certain class of problems. However, in real world optimisation problems it is hard to find the properties of the problem at hand in advance. Thus, we treat the problem as black-box. In black-box scenario, a method learns from the past performance of the parameter values and decides the suitable parameter value according to the current landscape.

We use the BBOB (Black-box optimisation benchmarking) [66] problem suite to train and test the proposed algorithms. BBOB provides an easy to use tool-chain for benchmarking black-box optimisation algorithms for continuous domains and to compare the performance of numerical black-box optimisation algorithms. It consists of 24 noiseless continuous benchmark functions [67] shown in Table 4.1. Each function consists of 15 different instances, totalling to 360 function instances. An instance of a function is a rotation and/or translation of the original function leading to a different global optimum. These 24 functions are grouped in five classes, namely, separable functions ($f01 - f05$), function with low or moderate conditioning ($f06 - f09$), functions with high conditioning and uni modal ($f10 - f14$), multi modal functions with adequate global structure ($f15 - f19$) and multi modal functions with weak global structure ($f20 - f24$).

TABLE 4.1: BBOB class and their functions

Class name	**Function name**
Separable functions	Sphere function ($f01$), Ellipsoidal Separable function ($f02$), Rastrigin Separable function ($f03$), Büche-Rastrigin function ($f04$), Linear Slope ($f05$)
Function with low or moderate conditioning	Attractive Sector function ($f06$), Step Ellipsoidal function ($f07$), Rosenbrock Original function ($f08$), Rosenbrock Rotated function ($f09$)
Functions with high conditioning and uni modal	Ellipsoidal non-Separable function ($f10$), Discus function ($f11$), Bent Cigar function ($f12$), Sharp Ridge function ($f13$), Different Powers function ($f14$)
Multi modal functions with adequate global structure	Rastrigin non-Separable function ($f15$), Weierstrass function ($f16$), Schaffers $f7$ function ($f17$), Schaffers $f7$ Moderately Ill-conditioned function ($f18$), Composite Griewank-Rosenbrock function $f8f2$ ($f19$)
Multi modal functions with weak global structure	Schwefel function ($f20$), Gallagher's Gaussian 101-me Peaks function ($f21$), Gallagher's Gaussian 21-hi Peaks function ($f22$), Katsuura function ($f23$), Lunacek bi-Rastrigin function ($f24$)

4.2 Offline tuning of hyper-parameters

Although we have contributed in the domain of parameter control, the adaptive methods in the first two chapters of contributions are combined with an offline tuning method. The offline and online optimising methods are present at different levels. Combining offline and online parameter setting allow us to utilise their best properties and at the same time balance the limitations of both approaches.

All the methods proposed for the selection of DE operators involve hyper-parameters. We have employed a well-known tuner known as *irace* (iterative racing) [115] to tune the hyper-parameters of the proposed adaptive methods within DE. In addition to tune the hyper-parameters of AOS methods, *irace* also tunes the other parameters of DE algorithm. *irace* is a racing algorithm that saves the hassle of manual tuning and allows for a fully specified and reproducible procedure. The input given to *irace* is the range of all parameters that need tuning and a set of training function instances. We have given *irace* a total budget of 10^4 evaluations for the tuning. *irace* starts tuning procedure by sampling a number of candidate parameter configurations from either a randomly initialised probability distribution or from a population of starting configurations given to it. The following steps are repeated until a budget given to *irace* is exhausted. The generated candidate configurations are evaluated on a sequence of problem instances. This process of evaluation on a sequence of instances is known as racing. During the racing process, poor performing candidate configurations are discarded and elite ones survive. A race terminates once the allocated computation budget is exhausted or the number of surviving candidate configurations is below some specific number. The best candidate configurations are then used to update the probability distribution. The distribution is biased towards good candidate configurations and is used to generate new candidate configurations for the next iteration.

Although all the proposed methods involve hyper-parameters, we utilise *irace* to tune AOS methods presented in Chapter 5 (*RecPM-AOS*) and Chapter 6 (*U-AOS-FW*). The method proposed in Chapter 7 (DE-DDQN) involves training a deep neural network whose weights are optimised using a gradient algorithm. An attempt has been made to tune the other hyper-parameters of DE-DDQN using *irace*. The

TABLE 4.2: Training set. $fxiy$ denotes a function instance iy that is obtained by a transformation of original function fx.

Function class	**Function instance**
Separable functions	$f01i01$, $f01i07$, $f02i09$, $f02i15$, $f03i10$, $f03i05$, $f04i08$, $f04i06$, $f05i07$, $f05i01$
Function with low or moderate conditioning	$f06i13$, $f06i07$, $f07i02$, $f07i05$, $f08i06$, $f08i03$, $f09i10$, $f09i03$
Functions with high conditioning and uni modal	$f10i11$, $f10i04$, $f11i09$, $f11i02$, $f12i01$, $f12i03$, $f13i13$, $f13i12$, $f14i12$, $f14i11$
Multi modal functions with adequate global structure	$f15i07$, $f15i15$, $f16i02$, $f16i14$, $f17i12$, $f17i15$, $f18i09$, $f18i15$, $f19i01$, $f19i09$
Multi modal functions with weak global structure	$f20i10$, $f20i06$, $f21i05$, $f21i11$, $f22i01$, $f22i08$, $f23i03$, $f23i15$, $f24i08$, $f24i04$

combined code of DE-DDQN with *irace* can be found on Github.[1] In the tuning phase, one setting of hyper-parameters takes more than a week to converge and it is estimated for *irace* to take atleast six months to return a final optimal configuration. Thus, we set the hyper-parameters of DE-DDQN as default values. We leave this part as future work where techniques like Bayesian Optimisation [123, 154] can be employed as a tuner for DE-DDQN.

4.2.1 Training set

The adaptive methods are proposed with the intention to generalise on different classes of functions as described in section 4.1. To prevent over-fitting, the training set contains two randomly selected function instances out of 15 from each of the 24 functions. Thus, training set consists of total 48 out of 360 function instances. The 48 function instances are shown in Table 4.2. The training set is common to all

[1]https://github.com/mudita11/Tune-DE-DDQN/

the presented algorithms. During training, each selected function instance from the training set is given a budget of 10^4 function evaluations on dimension 20.

Part III

Contributions

Chapter 5

Recursive Probability Matching

This chapter presents an adaptive operator selection (AOS) method in an attempt to improve the performance of DE by maximising the future reward attained with the application of an operator. It considers individual level selection of operators where one of the operator from a set of operators, *Op*, is selected for each parent in a generation. The presented AOS method is based on Markov Reward Process (MRP) [73] which consists of states, rewards and probability matrix (also known as transition probability). MRPs can be seen as a sequence of states where goal is to maximise the accumulated reward during the sampled sequence. Transition probability is a matrix that defines the probability of moving from one state to another assigning reward to each state in the process.

The AOS method presented is a variant of Probability Matching (*PM*) [57] named as Recursive Probability Matching (*RecPM*). Probability Matching has been initially proposed in the context of classifier system. It has later been used in EAs to map the performance of an operator to a probability and uses roulette-wheel selection to select an operator in the next generation [59]. *PM* probabilistically selects an operator according to its estimated quality. The quality of each operator is calculated as the weighted sum of a reward value, which measures the impact of the most recent application of the operator on solution fitness, and its historical quality. It is one of the most successful methods for AOS that is, for the online control of parameters in evolutionary algorithms.

In AOS, the algorithm needs to perform optimally from the current generation. This can be done by employing a mechanism that can maximise the future performance of the algorithm. Thus, in contrast to *PM*, *RecPM* estimates the quality of each operator according to a method inspired by MRPs from reinforcement learning [158]. MRPs employ Bellman equation which takes into account not only the reward values but also the selection probabilities of other operators. The goal in Bellman equation is to maximise the future cumulative reward. By combining *RecPM* with a credit assignment method based on offspring survival rate, we obtain the *RecPM-AOS* method.

5.1 Methodology

We present a novel AOS method inspired by Markov Reward Processes, which is used to predict the next state according to the expected reward given the current state. MRP is a framework from Reinforcement Learning that works in a stochastic environment. It assumes that the current state is independent of the whole history given the previous state known as Markov property. A state, S_{op} represents the selected operator op at a generation g and the corresponding reward is the immediate reward assigned to the operator $r'_{g+1,op}$, which is based on the impact of the application of the operator on the performance of the algorithm. Next we calculate the quality of each operator by adapting the Bellman equation [142, 158]. The Bellman equation is widely used to calculate the expected return starting from a state. Our motivation for using the Bellman equation is to use the historical performance of operators to predict their quality in the next iteration, which is then mapped to their probability of selection. Since the next operator is chosen probabilistically, we consider only transitions between states and rewards, and not actions.

We use the Bellman equation to estimate the quality $q_{g+1,op}$ of an operator op after its application in generation g as the expected value ($\mathrm{E}[\cdot]$) of its total sum of discounted future rewards:

$$q_{g+1,op} = \mathrm{E}[r'_{g+1,op} + \gamma r'_{g+2,op} + \cdots \mid Op_g = op] = \mathrm{E}[\sum_{z=0}^{\infty} \gamma^z r'_{g+z+1,op} \mid Op_g = op] \tag{5.1}$$

$$= \mathrm{E}[r'_{g+1,op} + \gamma \sum_{z=0}^{\infty} \gamma^z r'_{g+z+2,op} \mid Op_g = op] \qquad \text{(using recursive property)} \tag{5.2}$$

$$= r_{g+1,op} + \sum_{j=1}^{K} P_{op,j} \left[\gamma \, \mathrm{E} \left[\sum_{z=0}^{\infty} \gamma^z r'_{g+z+2,op} \mid Op_{g+1} = j \right] \right] \qquad \text{(assuming } \mathrm{E}[r'_{op}] = r_{op}\text{)} \tag{5.3}$$

$$= r_{g+1,op} + \gamma \sum_{j=1}^{K} P_{op,j} q_{g+2,j} \qquad \text{(using definition of } q_{g+2,op} \text{ in Eq. 5.1)} \tag{5.4}$$

where $r_{g+1,op}$ is the accumulated reward that stores all the past achievements for operator op and γ is the discount rate. It indicates the importance of the reward from the present onwards. A short-sighted algorithm assigns γ as a value of 0. That means, we only care about the current reward. If γ = 1, that means the far-sighted algorithms care about all the future rewards in full amount. A value of 0 considers the immediate success that does not provide information about the future and learns from a limited information. In contrast, a value of 1 can lead the accumulated reward value to explode. Thus, γ less than 1 ensures a finite reward value and it becomes important to find its optimal value. In the end of the Bellman derivation, it breaks down into two parts. Immediate reward value and discounted quality from next state weighted with probability matrix. In the context of AOS, we do not know the probability matrix P of size $K \times K$, thus we decided to calculate each entry as $P_{op,j} = p_{op} + p_j$, that is, as the sum of the selection probabilities of operators op and j. Here K denotes the total number of operators considered for adaptation. The rationale behind the formula above is as follows: When estimating $q_{g+1,op}$, operator op competes with all other operators $j \in Op$, including itself, since the selection of other operators in the past has impact on the current performance of the selected operator. Thus, their probabilities are added and multiplied by the quality estimate of operator j. These values are then aggregated in the end to get an overall estimate for operator op. The quality is an estimate not because of the expected values, which are assumed to be completely provided by the method, but because $q_{g+2,j}$ is not known and the current estimate at $g+1$ is used instead. When considering all operators, this forms a system of linear equations and can be re-written in the following vector form:

$$Q_{g+1} = R_{g+1} + \gamma P Q_{g+1} \quad \text{or} \quad Q = (1-\gamma P)^{-1} R \tag{5.5}$$

where $Q = [q_i]$ and $R = [r_i]$ are the K-dimensional quality and reward vectors that are updated at the end of each iteration g. The system of linear equations can be solved efficiently by matrix inversion [142] when the number of operators is small. Q is then normalised using the softmax function, which "squashes" each real value to a K-dimensional vector in the range $[0, 1]$ using the exponential function. Once the quality is estimated for each operator, the probability vector $p = [p_i]$ and probability matrix P are updated. The probability vector is updated according to equation 3.3 and used for the selection of an operator.

So far we have seen *RecPM* as an operator selector in AOS. It utilises the steps of Probability Matching except for the definition of operator quality, which is estimated using the Bellman equation as shown above. However, to obtain an AOS method, we still need to specify the credit assignment method that updates the reward values after the application of the selected operators at time step g. We propose to calculate the immediate reward $r'_{g+1,op}$ assigned to the selected operator op as the ratio of the number of offspring that survive to the next generation $g+1$ generated with the application of operator op, denotes as $N^{\text{surv}}_{g+1,op}$, to the population size NP. We define the accumulated reward $r_{g+1,op}$ assigned to an operator op as the immediate reward plus half the accumulated reward received in the previous generation. Any unselected operator receives half of the accumulated reward from the previous generation. Thus, each operator gets a fraction of last reward value, that stores its historical performance, and the selected ones get extra reward. Equation 5.6 shows the formulae for reward calculation where the value of 0.5 as weight assigned to $r_{g,op}$ is chosen by intuition.

$$r_{g+1,i} = \begin{cases} r'_{g+1,op} + 0.5 \cdot r_{g,op}, & \text{if } op \text{ is selected} \\ 0.5 \cdot r_{g,i}, & \forall i \neq op \end{cases}, \text{ where } r'_{g+1,op} = \frac{N^{\text{surv}}_{g,op}}{NP} \tag{5.6}$$

The rationale behind this credit assignment is that, if the operator is unlucky and not getting selected for enough number of generations, it still receives some reward based on its past performance and it has a chance of being selected in the future. This ensures that such operator is not discarded completely and may be selected after a certain number of generations.

As the algorithm progresses, the probability distribution of each operator is updated based on previous generation performance. The quality estimate $q_{g+1,op}$ of an operator op for generation $g+1$ is calculated as a weighted sum shown in Eq. 3.2.

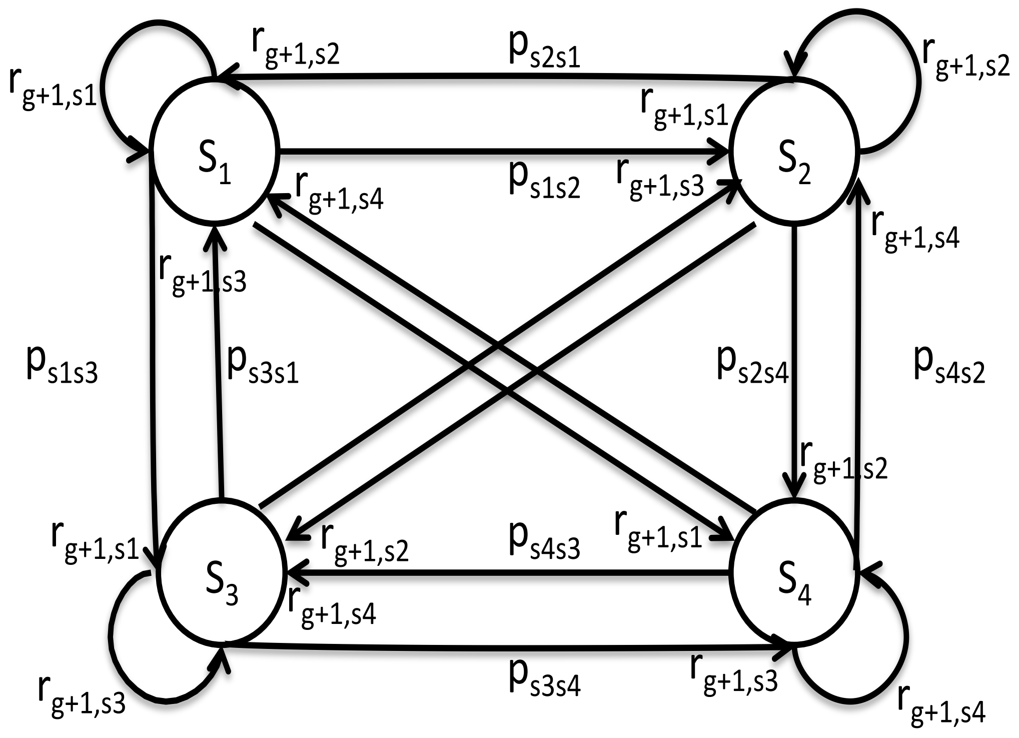
The combination of *RecPM* with the above credit assignment leads to a novel AOS method named *RecPM-AOS*. Figure 5.1 represents the working of *RecPM-AOS* mapped as markov reward process. For simplicity, the figure shows the working when population size is 1 that is there is only one parent to evolve. A state represents the selected operator in a generation g. In a state S_i, the operator application results in the assignment of reward $r_{g+1,si}$ to the operator. $r_{g+1,sj}$ represents a dynamic variable that captures the usefulness of an operator in state s_j depending on its successful application in a generation. The probability to move from one state S_i in generation g to another state S_j in generation $g+1$ is represented as p_{sjsi}. It determines the selection of next state in the next generation $g+1$. Thus, from a state s_i there are always four possibilities to move to any of the four states including s_i.

RecPM-AOS is integrated within DE, named *DE-RecPM-AOS*, to make DE more efficient by adaptively selecting, at run-time, a mutation strategy among the four mutation strategies. These strategies are “rand/1”, “rand-to-best/2”, “rand/2” and “current-to-rand/1”, shown in section 2.2.3. DE combined with *RecPM-AOS* has five parameters: three belong to DE, namely, scaling factor (F), population size (NP) and crossover rate (CR), while discount factor (γ), and minimum selection probability $(p_{\min})$ belong to *RecPM-AOS*.

5.2 Experimental Design

We compare the performance of proposed *DE-RecPM-AOS* within DE with two other algorithms, namely *DE-F-AUC* [49] and *PM-AdapSS-DE* [59], for the online selection of mutation strategies in DE. AOS methods learn the adaptation on four mutation

FIGURE 5.1: *RecPM* as Markov Reward Process



strategies. More advanced DE variants are available in the literature, however, we want to understand and analyse the impact of the various AOS methods without possible interactions with other adaptive components of those variants. Nonetheless, for the sake of completeness, we also compare our results with two state-of-the-art algorithms *JaDE* [169] and *CMA-ES* [64]. *JaDE* is a DE variant that uses a mutation strategy called "current-to-pbest" and adapts the crossover probability CR and mutation factor F using the values which proved to be useful in recent generations. *CMA-ES* is an evolution strategy that samples new candidate solutions from a multivariate Gaussian distribution and adapts its mean and covariance matrix.

5.2.1 Parameter tuning

We tune the hyper-parameters of the *DE-RecPM-AOS*, *DE-F-AUC* and *PM-AdapSS-DE* along with parameters of DE using the offline automatic configurator *irace* [115]. Table 5.1 shows the range of all parameters that need tuning. The parameter $p_{\min}$is upper bounded by 0.25 (Note that if $p_{\min}$ is set to its maximum possible value of 0.25, then all four operators' probabilities will necessarily be 0.25. If the value is tuned to a lower level, then probabilities can adapt.) because we consider four operators (K)

TABLE 5.1: Hyper-parameter choices given to *irace*. An interval (a, b] represents a set of numbers x satisfying $a < x \leq b$

Parameter Name	**Type**	**Range**	**Notes**
		DE parameters	
F	Real	[0.1, 2.0]	Mutation Rate
CR	Real	[0.1, 1.0]	Crossover Rate
NP	Integer	[50, 400]	Population Size
		Reward Choice parameters	
$\mathcal{W}$	Integer	[1, 200]	Size of window
		Quality Choice parameters	
C	Real	(0.0, 1.0)	Scaling Factor
α	Real	(0.01, 1.0)	Adaptation rate
γ	Real	[0.1, 1.0]	Discount rate
		Probability Choice parameters	
$p_{\min}$	Real	[0.0, 0.25)	Minimum selection probability

TABLE 5.2: Optimal parameter configurations selected from the range shown below the parameter name. The following prefix abbreviations are used: RecPM for DE-RecPM-AOS, AdapSS for PM-AdapSS-DE and F-AUC for DE-F-AUC. The symbol '-' in the table means that the parameter is not applicable to the AOS method.

Algorithm name	**F**	**NP**	**C**	α	$p_{\min}$	γ	**W**	**C**
RecPM1	0.47	168	0.98	-	0.17	0.75	-	-
RecPM2	0.5	200	1.0	-	0.11	0.46	-	-
RecPM3	0.5	200	1.0	-	0.0	0.6	-	-
AdapSS1	0.51	117	0.97	0.48	0.22		-	-
AdapSS2	0.5	200	1.0	0.86	0.04	-	-	-
AdapSS3	0.5	200	1.0	0.6	0.0	-	-	-
F-AUC1	0.24	96	0.55	-	-	-	31	0.14
F-AUC2	0.5	200	1.0	-	-	-	5	0.35
F-AUC3	0.5	200	1.0	-	-	-	50	0.5

and $K * p_{\min} < 1$ to prevent probability to become negative. This is given as input to *irace*.

The training set consists of 48 of total 360 BBOB noiseless function instances [66], randomly selected within each class, to avoid over-fitting. For the working of *irace* and details on BBOB test suit and training set refer to Chapter 4. The AOS method with a fix configuration is run on a selected function instance to a maximum number of $10^4 \cdot n$ function evaluations (FEvals), where n is the dimension of the benchmark function. In this chapter, we focus on $n = 20$ for all functions.

In order to evaluate the impact of parameter tuning, we consider three parameter configurations of each algorithm. The first configuration is obtained by tuning

Algorithm 4 Differential Evolution with an AOS

```
1:  Initialise parameter values of DE ($F$, $NP$, $CR$) and AOS method
2:  Initialise and evaluate fitness of each individual $x_i$ in the population
3:  $g = 0$ (generation number or time step)
4:  while stopping condition is not satisfied do
5:      for each $x_i$, $i = 1, \dots, NP$ do
6:          if one or more operators not yet applied then
7:              $k$ = Uniform selection among operator(s) not yet applied
8:          else
9:              $k$ = Select mutation strategy based on selection method (AOS)
10:         Generate offspring using selected operator $k$
11:     Evaluate offspring population
12:     Perform survival selection
13:     Perform credit assignment (AOS)
14:     Estimate quality for each operator (AOS)
15:     Update selection value (eg. probability) for each operator (AOS)
16:     $g = g + 1$
```

all parameters of DE and the AOS methods. The second configuration is obtained by tuning only the parameters of the AOS methods, while the parameter values of DE are taken from [48]: $CR = 1.0$, $F = 0.5$ and $NP = 200$. The value $CR = 1.0$ means that a mutation strategy is applied to each dimension of all parents, which maximizes the impact of the mutation strategies. Finally, the third configuration (*default*) uses the settings suggested in [48] for *DE-F-AUC* and *PM-AdapSS-DE*, which uses the DE settings described earlier and AOS settings tuned with a different configurator, *F-Race*. All parameter configurations are shown in Table 5.2.

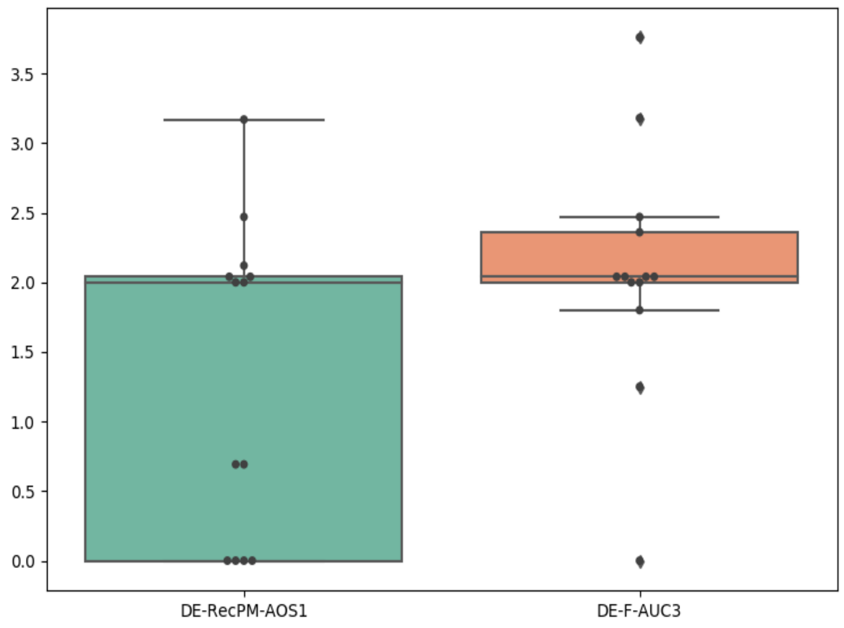
5.2.2 Testing phase

After tuning, each obtained configuration is evaluated on the remaining 312 function instances of the BBOB benchmark set. The AOS methods within DE are run to a budget of $10^5 \cdot n$ FEvals. The hyper-parameters of *RecPM-AOS* returned by *irace* are kept constant during the whole run of DE algorithm while *RecPM-AOS* adapts the operators of DE. The tuned *RecPM-AOS* within DE tested on a function instance is shown in Algorithm 4. In the initial generation, a population of candidate solutions are generated and evaluated. After this, the steps of *RecPM-AOS* method are repeatedly applied until a stopping criterion is satisfied. We have used two stopping conditions for DE either of which once satisfied, terminates the algorithm returning the best so far fitness value. First condition is the the fix function evaluations (budget) and the other is the target value, 10^{-8}, is reached. Each parent is evaluated using an operator selected using proportionate selection. Selection mechanism is applied to each parent and an offspring is produced. Either parent or offspring survive to become part of the parent population in the next generation based on the fitness evaluation. Accumulated reward is assigned to each operator followed by quality estimation and probability calculation. The fitness of elite solution is returned once the algorithm is terminated.

We use plots of the Empirical Cumulative Distribution Function (ECDF) to assess their performance (Fig. 5.3). These plots are auto-generated by BBOB benchmarking tool. The ECDF displays the proportion of problems solved within a specified budget of function evaluations (FEvals) for different targets $f_{\text{target}} = f_{\text{opt}} + \Delta f$, where f_{opt} is an the optimum function value to reach with some precision $\Delta f \in [10^{-8}, 10^{2}]$. In the plots, FEvals is given on the x-axis and y-axis represents the fraction of problems solved. ECDF for each function is the average performance of 13 function instances (one run per problem instances of function). To differentiate between different instances of a function we show box plots in Fig. 5.2. It is a result of average performance of 13 runs, each on different test instance of function $f21$. The algorithms *DE-RecPM-AOS* and *DE-FAUC3* in this figure are terminated when either the optima value is reached or the difference of best fitness value with optima is less than 10^{-8}. The optima value of different instances of a function can be different.

A large symbol '$\times$' in Figure 5.3 shows the maximum number of function evaluations given to each algorithm, in our case, $10^5 \cdot n$ FEvals are given to each algorithm with AOS method. Results reported after this symbol use bootstrapping to estimate the number of evaluations to reach a specific target for a problem [38], that is not necessarily reliable. The results denoted with `best 2009` correspond to the artificial best algorithm from the BBOB-2009 workshop constructed from the data of the algorithm with the smallest aRT (average Run Time) for each set of problems with the same function, dimension and target. The aRT is calculated as the ratio of the number of function evaluations for reaching the target value over successful runs (or trials), plus the maximum number of evaluations for unsuccessful runs, divided by the number of successful trials. Data to generate ECDF graphs for *DE-F-AUC3*,

FIGURE 5.2: Function error values obtained by 13 runs of *DE-RecPM-AOS* 1 and *DE-F-AUC3* on function $f21$



PM-AdapSS-DE3, *CMA-ES* and *JaDE* is obtained directly from the COCO website.[1] The trials that reached f_{target} within the specified budget are termed as successful trials, *#succ*. The aRT tables are shown in Appendix C.

We expected to tune *DE-F-AUC* and *PM-AdapSS-DE* algorithms with the hope to replicate the original results for *DE-F-AUC3* and *PM-AdapSS-DE3* [49, 59]. But we could not match the results shown in these papers. Thus, we decided to use the data available online at the COCO website and compare variants of proposed algorithm with *DE-F-AUC3* and *PM-AdapSS-DE3* only. The interested reader is referred to the supplementary material [151] to find the results of tuned *DE-F-AUC* and *PM-AdapSS-DE* algorithms. The ECDF graphs of variants of the proposed algorithm with *DE-F-AUC3* and *PM-AdapSS3* are shown in Figure 5.3 that show the performance of algorithms on 24 function instances and averaged performance over the tested functions. It also shows ECDFs of five problem classes. From now on in this chapter we only talk about the original results and not the replicated ones.

[1]`http://coco.gforge.inria.fr/doku.php?id=algorithms-bbob`

5.3 Experiments and Results

In this section we present the discussion on experiments conducted on BBOB problem set. We start by pointing out the performance differences within three variants of *RecPM-AOS*. These three variants differ in their hyper-parameter setting. Further the best variant of *RecPM-AOS* is compared with AOS and non-AOS algorithms.

5.3.1 Comparison of AOS methods with different parameter settings

The results obtained for three variants of *DE-RecPM-AOS* are as expected. The proposed algorithm with all tuned parameters outperformed its all other variants both in terms of speed and percentage of problems solved. When all three AOS methods use the default settings, it is estimated that *F-AUC* and *RecPM-AOS* solves the same number of problems but within the given budget all algorithms solved the same number of problems with varied speed. The case where only parameters of AOS method are tuned in proposed algorithm shows that *DE-F-AUC3* and *PM-AdapSS-DE3* solve maximum problems with almost same speed within the given budget. But when given more FEvals, according to bootstrapping technique, *DE-RecPM-AOS2* shows the same performance as *DE-F-AUC3* by solving the same number of problems whereas *PM-AdapSS-DE3* could not match the performance of other two algorithms. The proposed method with all tuned parameters that is, parameters of DE algorithm and of *RecPM-AOS* method outperformed all other algorithms by solving 75% of the problems. This is clearly because of the properties the proposed AOS method has. The tuned configurations of replicated algorithms: *DE-F-AUC* and *PM-AdapSS-DE* are not better than the original results reported, which we cannot replicate.

Summing up the above discussion, it can be said that tuning all the parameters of the proposed algorithm (*DE-RecPM-AOS1*) outperformed all its variants, thus tuning on training set plays an important role. It also outperformed all other AOS methods within DE solving 75% of the total problems. Thus, historical information preserving property in the form of reward and using Bellman equation to estimate quality of operator led to efficient adaptability of operators. On the other hand both F-AUC and RecPM-AOS make use of past performance of operators, we do that by defining reward of each operator capturing a fraction of its last reward which reduces the hassle of maintaining a window of certain size. However, *F-AUC* and *PM-AdapSS* show similar speed in solving a fixed number of problems and *DE-RecPM-AOS1* has faster convergence speed and increased percentage of problems solved.

5.3.2 Comparison of *RecPM-AOS* with state-of-the-art algorithms

CMA-ES and *JaDE* are given a budget of $5 \cdot 10^4$ FEvals. When comparing different versions of *DE-RecPM-AOS* with *CMA-ES* and *JaDE*, the proposed algorithm with all tuned parameters is able to solve more functions than *CMA-ES* as seen in the overall ECDF graph shown in Figure 5.3 that is, almost 10% more than the best variant of *DE-RecPM-AOS*: *DE-RecPM-AOS1*. However, *JaDE* manages to solve majority of the problems compared to other AOS methods within DE. In the initial runs, *CMA-ES* has faster convergence speed than any other algorithm.

We want to understand the selection of operators that led to *RecPM-AOS* perform better or worse than other algorithms in comparison. Thus, we randomly select an instance among 13 test function instances of functions $f05$, $f03$ and $f07$. Figure 5.4 shows graphs for instance 1 of each function 5, 3 and 7, denoted as f05 i01, f03 i01 and f07 i01 respectively. A value of $p_{\min} = 0.17$ controls the level of adaptation of operators. *RecPM-AOS* could not solve many problems for function 3 instance 1 but is ranked second among all algorithms in comparison. It reaches all targets for f07 and f05 i01 with competitive speed for only the former function. Thus we selected function instances with mixed performance. For each instance, we show two figures: the first figure shows the selection of operators and the second figure demonstrates the improvement of the best solution in a run. The first figure shows a series of horizontal bars representing the use of different operators in different generations. An operator can be identified with a unique color. Each bar raise to the same level indicating the number of selection made in a generation. That is, a total of 168 selections are made, equal to population size, to evolve 168 parents in each generation. For instance the first horizontal bar for function 5 shows that near-proportionate use of all four operators in the first generation and last bar represents last generation with operator 3 (“rand-to-best/2”) being most selected in comparison to any other operator.

FIGURE 5.3: ECDFs on test set

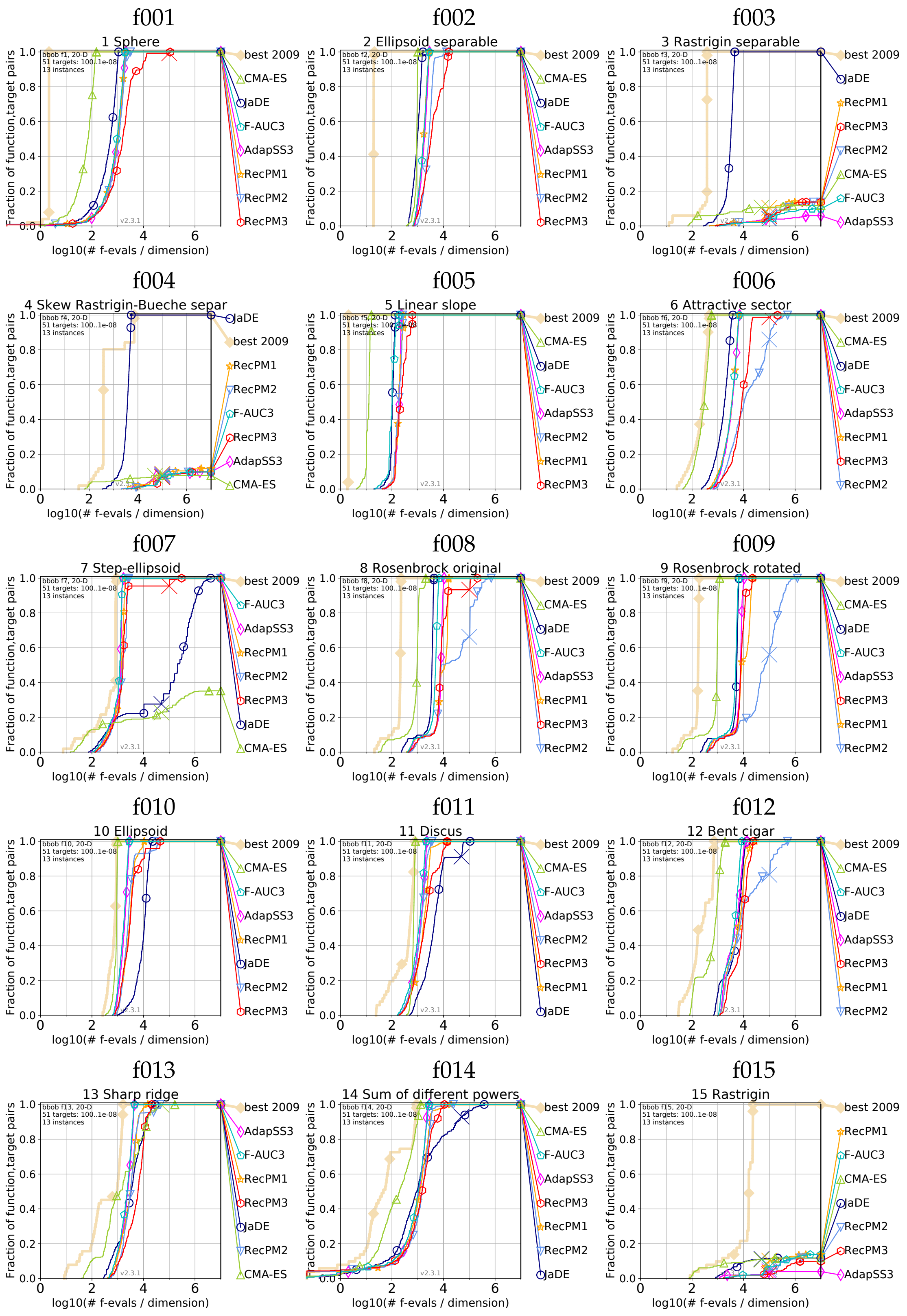
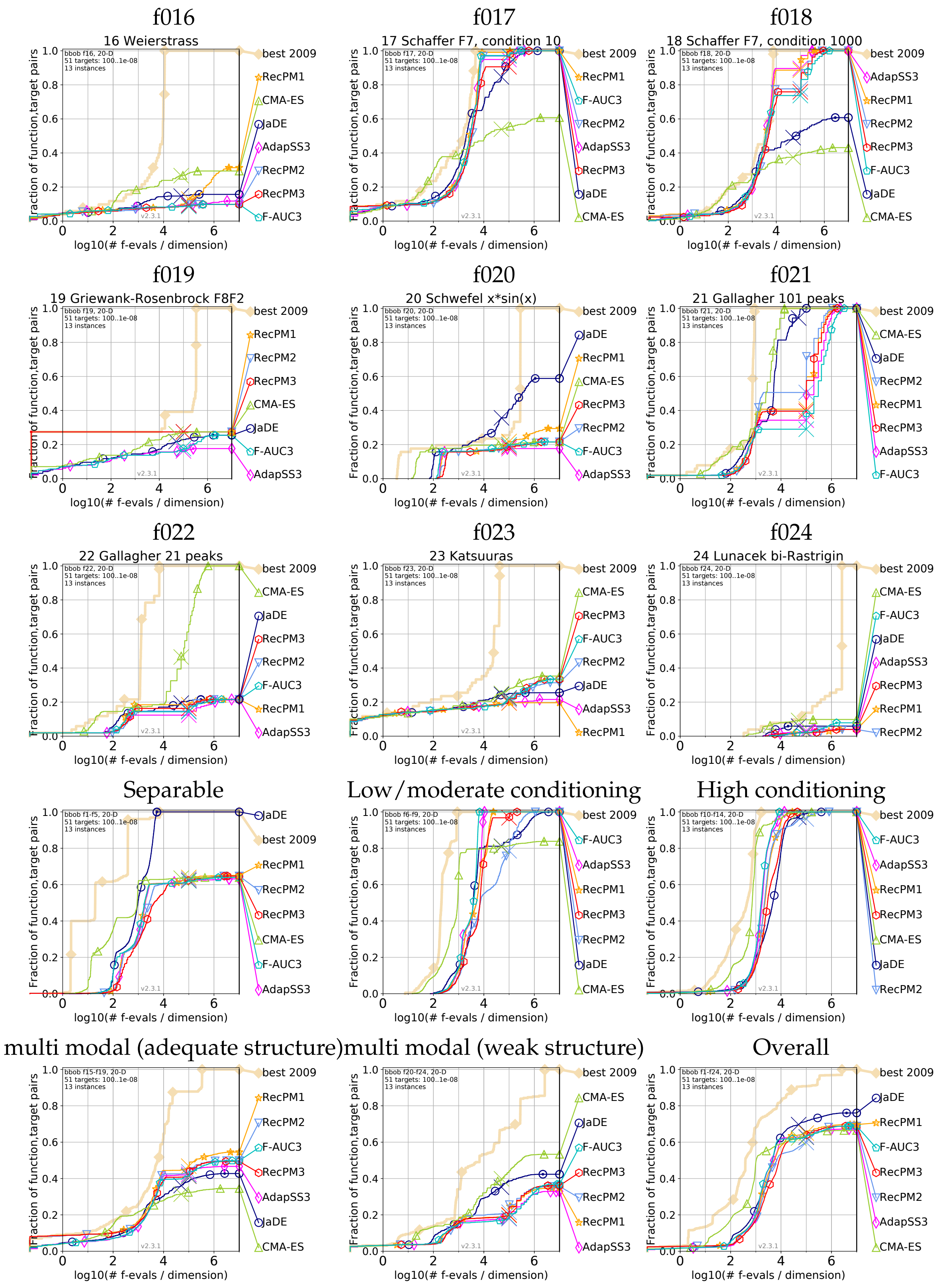


FIGURE 5.3 (cont.): ECDFs on test set



The second figure for each function shows the best fitness in different generations for a run. The operator selection graph for function 3 and function 7 looks tightly packed in comparison to function 5 because the optimum value is not attained even when all the budget is exhausted and the algorithm is run as long as the stopping criterion is not satisfied. In contrast for function 5, all targets achieved in generation 18 and algorithm stops before exhausting all the budget.

RecPM-AOS chose to utilise rand-to-best/2 operator to optimise $f05i01$. However, it lacked in speed with majority of the algorithm in comparison. $f05$ is a linear function and *RecPM-AOS* reaches its optimum within the given budget and in 18 generations. rand-to-best/2 combines current and best solution making the offspring fall in the direction of best or current solution. Limited application of rand/1 help maintain diversity. In both $f03i01$ and $f07i01$, *RecPM-AOS* decides to select operator rand-to-best/2 most of the times, rand/2 and rand-to-best/2 almost proportionately and rand/1 is applied least. This pattern is seen throughout the generations. This selection has proved beneficial for function $f07$ but not $f03$. $f07$ reaches all targets exhausting all the budget. The difference in their performance could be due to their modality. Although $f07$ is non-separable with many plateaus of different sizes, it is uni modal with one optimum, whereas $f03$ is separable and multi modal. The use of each operator with most focus on exploiting current and best candidates has proved helpful for $f07$ to avoid premature convergence.

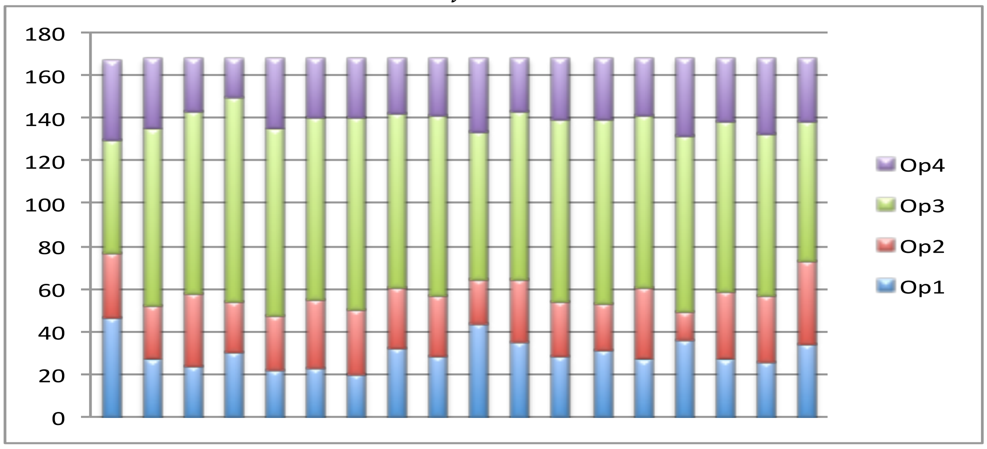
5.4 Summary

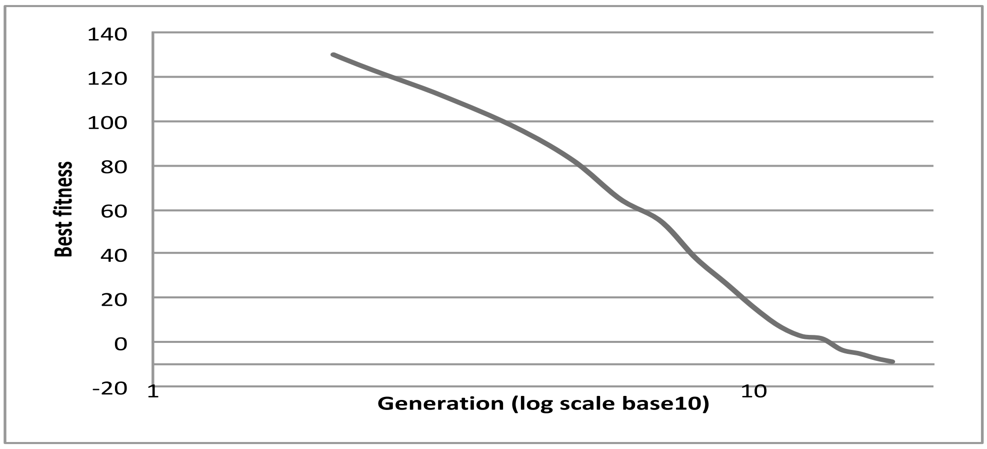
We presented a variant of probability matching, *RecPM-AOS*, as a parameter control method that assigns the quality of an operator as an aggregated estimate of future performances of operators and estimates reward based on progress in past generations. The proposed algorithm is a *PM* variant called Recursive Probability Matching. The main difference between *RecPM* and classical *PM* is in the way the latter assigns the quality to a strategy.

The proposed AOS method is applied to the online selection of mutation strategies in DE on the BBOB benchmark functions. The new method is compared with two AOS methods, namely, PM-AdapSS, which utilises probability matching with

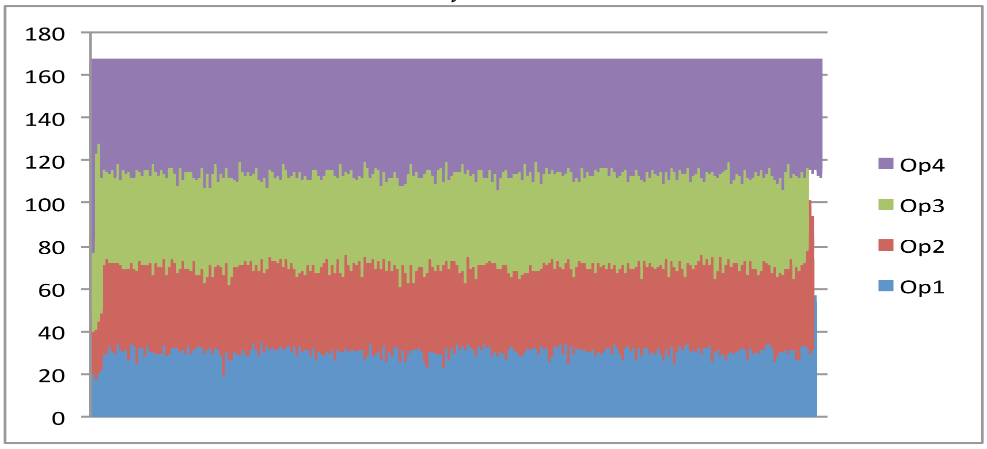
FIGURE 5.4: Operator application and best fitness graphs for *DE-RecPM-AOS* ($p_{\min}$= 0.17). Op1: "rand/1", Op2: "rand/2", Op3: "rand-to-best/2", Op4: "current-to-rand/1"

*f*05*i*01





*f*03*i*01



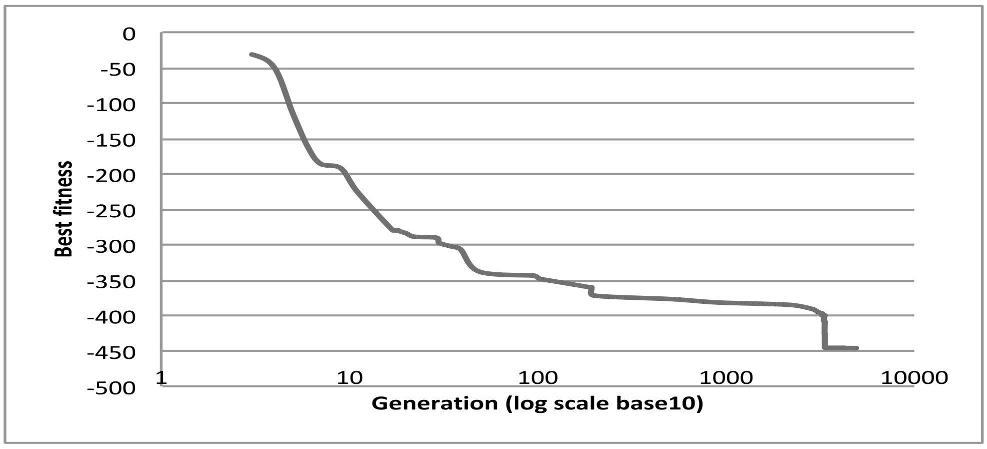
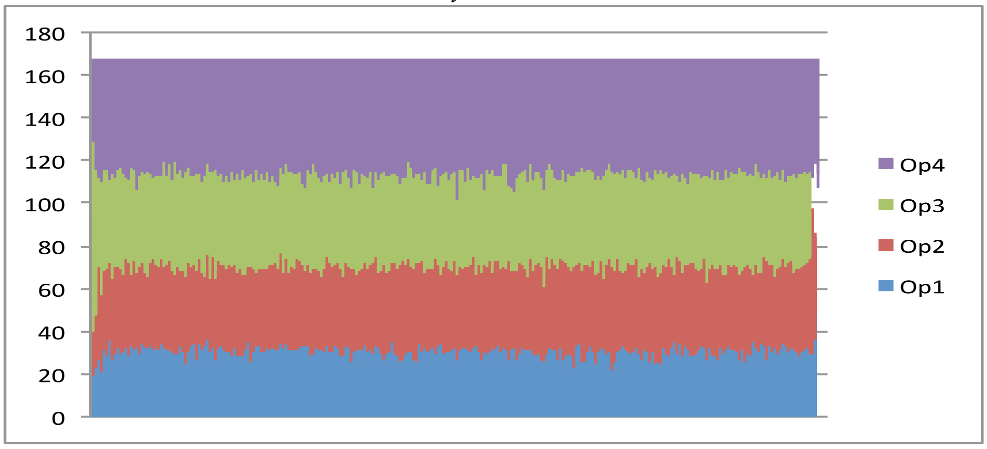
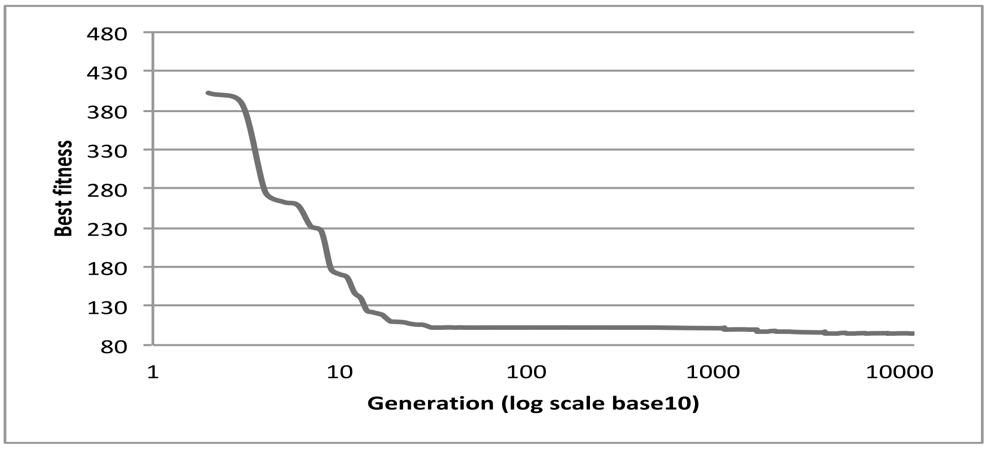


FIGURE 5.4 (cont.): Operator application and best fitness graphs for *DE-RecPM-AOS* (p_{min}= 0.17). Op1: "rand/1", Op2: "rand/2", Op3: "rand-to-best/2", Op4: "current-to-rand/1"

*f*07*i*01





relative fitness improvement, and F-AUC, which combines the concept of area under the curve with a multi-arm bandit algorithm. It is also compared with two non-AOS state-of-the-art algorithms *CMA-ES* and *JaDE*. *irace* is used to find a good offline settings for the proposed AOS method, which illustrates the usefulness of offline procedures to successfully design new online adaptation methods. It is used to train the parameters on approximately 48 of the total 360 function instances.

Experimental results show that the tuned *RecPM-AOS* method is the most effective at identifying the best mutation strategy to be used by DE in solving most functions in bbob among the AOS methods. *Recursive-PM* within DE with tuned parameters shows that it outperforms the other two AOS methods, *DE-F-AUC* and *PM-AdapSS-DE*, and *CMA-ES* by solving 75% of the problems. The proposed algorithm could not outperform *JaDE*, but had similar convergence rate.

Chapter 6

Unified Framework for Adaptive Operator Selection

In the previous chapter, we presented a variant of Probability Matching known as Recursive Probability Matching (*RecPM-AOS*). *RecPM-AOS* is an improved AOS method that aims to maximise future cumulative reward with the help of the Bellman equation in markov reward processes. In this chapter, we extend the idea of AOS by introducing new AOS methods in a unifying framework. This procedure introduces a number of hyper-parameters such as the fraction of previous accumulated reward (a weight) to take under consideration when designing the reward definition in *RecPM-AOS*. We employ a tuner to select an AOS method and to tune its hyper-parameters.

As seen in chapter 3.1, AOS has mainly two components, Credit Assignment (CA) and Operator Selection (OS). CA involves a definition based on the fitness achievement over a solution. The most commonly used CA technique is fitness improvement from parent to offspring. OS takes the information captured by CA and estimates the quality of each operator followed by calculating its probability. This is followed by selection and probability techniques. For comprehensive literature on AOS, refer to chapter 3.1.

Many novel AOS methods can be designed by combining different components of existing AOS methods. To test the efficiency of these methods as an AOS method, this chapter presents a unified AOS framework that builds upon the existing classification of an AOS method [47]. This is done by analysing multiple AOS methods from the literature to design a simplified framework. The AOS methods used to build the framework are originally proposed in the literature to tune the parameters of various EAs such as genetic algorithms and differential evolution. The framework consist of multiple choices for each AOS component. Some of the choices are inspired from Reinforcement Learning utilised to adapt parameters of different EAs. The framework can be utilised to explore different combinations of the AOS components' choices. As an estimate, we can generate more than 5000 novel AOS methods from the framework. It can also be used to replicate various known AOS methods from the literature. An AOS method is build from the framework by setting the component choices and fixing the values of their parameters. In order to make the framework widely applicable, choices with diverse properties are included such as immediate progress to far-sighted progress, focusing on the clustered achievement to the outliers, etc. In addition to this, some novel choices are added to the components and each choice is generalised. In the process of generalisation, a number of hyper-parameters are introduced within the choices. The framework is applicable for the online adaptation of discrete parameters of an evolutionary algorithm.

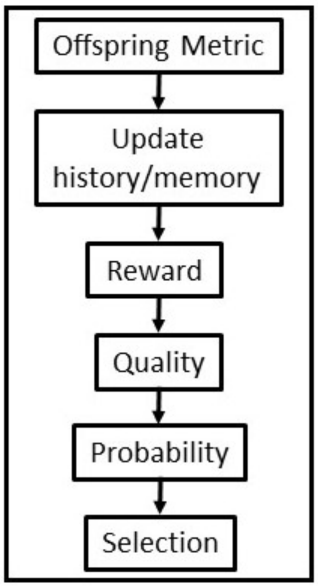
As the framework consists of various AOS methods with their hyper-parameters, an offline configurator is employed to select an optimal AOS method and tune its hyper-parameters. Thus, we present a combination of the framework consisting of online adaptive methods with an offline configurator to improve the search performance of differential evolution (DE). Along with selecting an AOS method with its parameters, the tuner also decides the parameters of the DE algorithm. We have utilised the framework to find a suitable tuned AOS method to adapt nine commonly used mutation strategies in DE on the BBOB benchmark set. The resultant framework is flexible enough to replace DE with any EA to tune its discrete set of parameters.

6.1 Methodology

The following three tasks are performed to achieve a unified framework of AOS methods combined with an offline configurator:

- We build upon the existing classification [35] by identifying the new components to simplify the structure of AOS. AOS methods are known to have two

FIGURE 6.1: Adaptive Operator Selection components

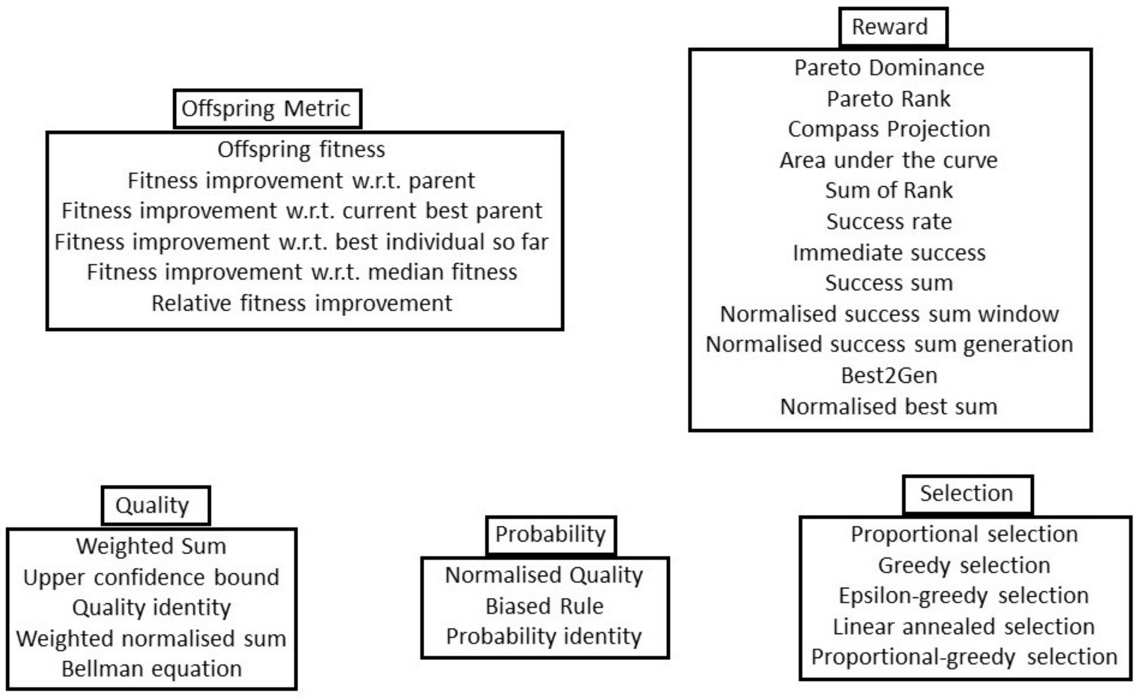


major components, credit assignment (assigning reward to an operator) and operator selection (assigning probability to each operator based on quality). We have used this existing classification and further classify these components. The classification is shown in Figure 6.1

- A simplified taxonomy is represented that consists of five components with different heuristics as their choices. Thus, the framework consists of an in-depth formulation of AOS components with a generalised structure. Each AOS component with its choices is shown in Figure 6.2

- The resultant framework consists of various AOS designs, out of which one needs to be selected to perform online tuning of parameters in an EA. Thus, a tuner (a meta-leaner) is employed to find a near optimal configuration setting or combination of choices for a given set problems. A well-known offline configurator known as *irace* is used for this purpose. The selected framework has its own parameters to be tuned which do not directly impact the problem solution space. Thus, an offline configurator can be utilised efficiently to give a static value for these hyper-parameters. To tune these hyper-parameters we employ the same tuner in combination with the framework. The role of *irace* is to offline select a combination of component choices given a set of problems

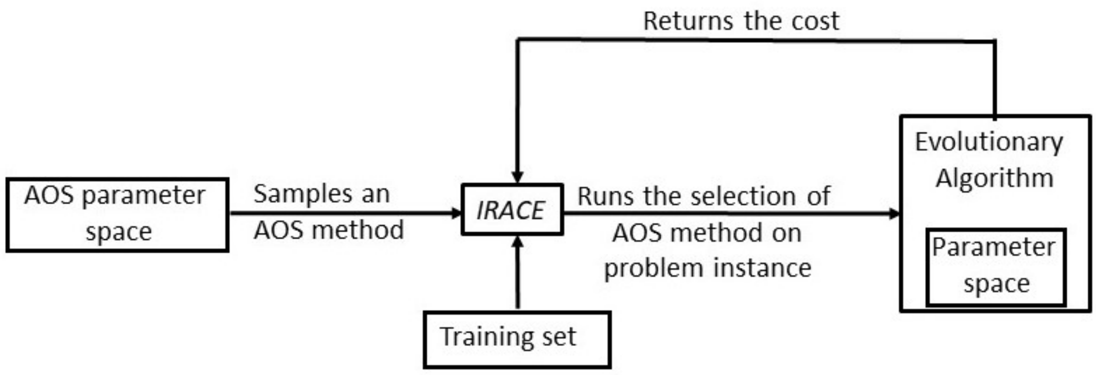
FIGURE 6.2: Adaptive Operator Selection component choices



and tune the hyper-parameters of the selected AOS method along with the parameters of DE.

Figure 6.3 shows the simplified training procedure of the framework using an offline tuner *irace*. *irace* samples a configuration from the component space and its hyper-parameter space related to the selected choices. It also samples a configuration from the DE parameter space. These choices remain static while the algorithm runs on a selected problem instance selected from a training set. During this run, the AOS method formed by component choices selected by *irace* online tunes the mutation strategy of DE. At the end of an EA run, the best seen fitness value is sent to the *irace* in the form of cost to make an informed decision on the optimal choices given the training set. This is done repeatedly until a budget given to *irace* is exhausted and the configuration that performed best is returned. This configuration consists of a choice from each AOS component, its tuned hyper-parameters and parameter values of DE. This describes a single step of *irace*. For full working of *irace* refer to Chapter 4.

FIGURE 6.3: Unified Adaptive Operator Selection architecture



6.2 Components of the proposed framework for AOS

In this section the components of the proposed framework are discussed in detail. It consists of five components each with a number of generalised choices. The five components are offspring metric (*OM*), reward ($r_{g+1,op}$), quality ($q_{g+1,op}$), probability ($p_{g+1,op}$) assignment and selection mechanism (op_{g+1}). We consider individual level control that is in a generation, an operator is selected for each parent. At the end of the generation, *OM* assigns a value to each offspring according to the improvement gained with the operator application. To prepare the selection of operators for the population in the next generation, reward, quality and probability of each operator are updated, according to the *OM* values. In the end, based on the probability values of each operator, the selection method is used to select the operator for each parent to produce offspring. Further we discuss each of the component choices one by one in detail.

6.2.1 Offspring Metric

We define an *OM* as a function of some statistics on population fitness. The metric, mathematically represented as ($OM(g, k, op)$), assigns a k-th improved value to the i-th improved solution (offspring) $x_{i,op}$ generated after using operator *op*. If there is no improvement, 0.0 metric value is assigned to the offspring. That implies that the offspring ($OM(g, k, op)$) is as good as its parent. The value depends on the parent fitness, the offspring fitness and other significant references shown in Table 6.1. The table shows six offspring metrics all of which are designed to be maximised when the objective function is a minimisation problem. That is, for an $OM(g, k, op)$, the higher the value of i-th offspring, the better the offspring is.

We store all *OM* in memory that can be of two types, generation memory and window memory. The generation memory stores six *OM* values for each offspring in each generation. These values are shown in Fig. 6.1. As the algorithm progresses, the size of generation memory grows. A window memory of size $\mathcal{W}$ is formed using the generation memory. Each entry in the window consists of six values resulting from *OM*. The window memory stores these six metric values of an offspring only if it improves over its parent. In other words, it stores a finite set of *OM* generated by any operator. Initially, the window is filled as the offspring are generated. Once the window is filled, the new improved offspring generated by an operator is inserted in First In First Out (FIFO) manner such that the offspring from the window generated by the same operator to enter first is removed and new offspring metric data is put at top of the window. If there is no application of that operator present in the window, the worst offspring data is removed. The generation memory and window memory are updated at the end of each generation. The data from generation memory can be utilised either for fix number of generations or for fix number of operator applications described below:

- Dynamic number of operator applications (max_{gen} as a parameter): The *OM* values produced by an operator in the last fix number of max_{gen} generation(s) are taken into account. It is important to note that the number of applications of each operator can vary in each generation. Thus, in max_{gen} number of generations the number of total applications of an operator can be different from others.

- Fix number of operator applications (fix_{appl} as parameter): In this case the last fixed number of operator applications are considered. The generation span depends on the improved offspring in each generation and is not known in advance.

Equation 6.1 [113, 78, 143, 99, 141, 21, 100] defines the offspring fitness. For the minimisation problem, this metric assigns the offspring an *OM* as the negative of

TABLE 6.1: Offspring Metrics ($OM(g,k,op)$)

Name	Definition
Offspring fitness	$-f(u_{i,op})$ (6.1)
Fitness improvement w.r.t. parent	$\max\{0, f(x_i) - f(u_{i,op})\}$ (6.2)
Fitness improvement w.r.t. current best parent	$\max\{0, f_{best} - f(u_{i,op})\}$ (6.3)
Fitness improvement w.r.t. best individual so far	$\max\{0, f_{bsf} - f(u_{i,op})\}$ (6.4)
Fitness improvement w.r.t. median fitness	$\max\{0, f_{median} - f(u_{i,op})\}$ (6.5)
Relative fitness improvement	$\frac{f_{bsf}}{f(u_{i,op})} \cdot \max\{0, (f(x_i) - f(u_{i,op}))\}$ (6.6)

the raw fitness. In case of the maximisation problem, the raw fitness value should be considered.

The next four equations 6.2 – 6.5 define the OM as the difference between the fitness of an offspring generated by the application of operator op and a reference point. The reference points considered in this study are parent fitness ($f(x_i)$) [78, 72, 77, 50, 47, 131, 26, 113, 15, 120, 117, 164, 49, 118, 3], best parent fitness (f_{best}) [78, 77, 31], best individual fitness so far (f_{bsf}) and median population fitness (f_{median}) [86, 85, 84]. Getting a value 0 shows that there is no significant improvement in the offspring from a reference point. These four metrics follow the following rule: the farther the offspring from the reference point, the higher the OM will be.

The offspring fitness improvement shown in equation 6.2 has been most widely used in the literature showing its significance in giving useful information of an operator. Among these four metrics, we propose to include the best seen candidate as a reference point shown in equation 6.4. It is an important reference as it gives a search direction for the exploitation in the neighborhood of the best so far candidate. In this case, an operator that produce an offspring with higher fitness in reference to the best so far candidate gets higher reward value compared to the operator producing offspring with relatively lower improvement.

Relative fitness improvement defined in equation 6.6 was originally proposed in [127] and later used in [59] as part of an AOS method. It takes into account the fitness improvement from parent to offspring along with the best so far candidate fitness.

6.2.2 Reward

The reward given to an operator *op* at generation $g + 1$, represented as $r_{g+1,op}$, gives a measure of achievement of an operator. It goes beyond the current performance of an operator using either generation or memory window. In a run of EA, it is a function of one of the selected *OM*. The reward assigned to an operator is maximised as the *OM* is designed to be maximised. That means if an operator has performed better than another operator, then former should get higher reward compared to the latter for any selected *OM*.

We present a generalised classification of the reward definitions from the literature shown in Table 6.2. They utilise an *OM* definition for an operator in a specifically defined manner. Some make the use of direct fitness values such as diversity and quality, weighted fitness average and best *OM*; others using fitness based score such as ranking and count of improved *OM*. Reward combines one or two of these statistics, thus learning from limited amount of information available from the fitness landscape.

Some authors consider rewarding ancestors of a well-performing operator [166, 15] in addition to rewarding the current operator itself; others do not consider such a case [113]. By ancestor we mean the operators that lead to the good performance of current operator. We decided not to include that option in the classification as [15] suggests that it sometimes degrades the results. In the framework, any method that involves clock time [143] is replaced by function evaluations in some cases. The framework updates reward at the end of each generation that is contrary to the methods that chose to update reward after a certain number of generations [134]. This implies that there is no update on AOS components for few generations because this can lead to loss of information. [117] introduces the idea of removing and adding operators from a storage called credit registry in an AOS. We do not maintain such a registry in the framework with the intuition that eventually an AOS can learn to select the optimal operator from a list of operators depending on the current stage of an EA.

The reward choices are divided into the following five categories: fitness diversity and quality, comparison based, successful operator applications, fitness sum and best offspring. The choices under these categories share similar properties. Fitness diversity and quality reward choices are made with the selected *OM* diversity and quality for a fix number of applications, choices under comparison based category involve ranking the *OM* from the window memory, successful operator applications comprise of the number of *OM* resulting from successful operator application, choices in fitness sum simply add the raw *OM* values and lastly best offspring category consists of choices formed with the best solution fitness in a generation combined from a certain number of generations.

Fitness Diversity and Quality This category includes the diversity (standard deviation) and quality (average) over an *OM*, combined in different manner. The three definitions consider fix number of operator applications, fix_{appl}, extracted from generation memory. These two statistics are represented as a coordinate ($o_{op}^{fix_{\text{appl}}}$) in two dimensional space for each operator *op* shown in the equation below:

$$o_{op}^{fix_{\text{appl}}} = (div_{i=k}^{k-(fix_{\text{appl}}-1)} OM(g,i,op),\ qual_{i=k}^{k-(fix_{\text{appl}}-1)} OM(g,i,op)) \tag{6.7}$$

The diversity *div* and quality *qual* of *OM* for an operator *op* are calculated for last fix_{appl} number of applications.

Pareto Dominance (PD) on ($o_{op}^{fix_{\text{appl}}}$) shown in equation 6.8 [117] counts the number of operators that are dominated by *op*. It is normalised by the sum of *k* operators' reward values. The best operator corresponds to the highest value of PD indicating that it generated maximum number of offspring in last certain number of generations compared to other operators.

Pareto Rank (PR) on ($o_{op}^{fix_{\text{appl}}}$) shown in equation 6.9 [117] method counts the number of operators that dominate *op*. The operator with the least PR value is the best of all operators. Operators with a PR value of 0 belong to the Pareto frontier. Both PD

TABLE 6.2: Reward ($r_{g+1,op}$)

Definition	Parameters
Fitness Diversity and Quality	
Pareto Dominance $\frac{\text{PD }(o_{op}^{fix_{\text{appl}}})}{\sum_{j=1}^{K} \text{PD }(o_{j}^{fix_{\text{appl}}})}$ (6.8)	fix_{appl}
Pareto Rank $\frac{\text{PR }(o_{op}^{fix_{\text{appl}}})}{\sum_{j=1}^{K} \text{PR }(o_{j}^{fix_{\text{appl}}})}$ (6.9)	fix_{appl}
Compass Projection $\lvert o_{op}^{fix_{\text{appl}}}\rvert \cdot \cos(\alpha_{op}) - \min_{j=1}^{K} \lvert o_{op}^{fix_{\text{appl}}}\rvert \cdot \cos(\alpha_{j})$ (6.10)	θ, fix_{appl}
Comparison (Rank)	
Area Under the Curve. Area under the curve based on the rank of $OM(g,k,op)$ (6.11)	D,$\mathcal{W}$
Sum of Rank $\frac{\sum_{r(OM(g,k,op))} D^{r(OM(g,k,op))}(\mathcal{W} - r(OM(g,k,op)))}{\sum_{r=1}^{\mathcal{W}} D^{r(OM(g,k,op))}(\mathcal{W} - r(OM(g,k,op)))}$ (6.12)	D,$\mathcal{W}$
Successful operator applications	
Success Rate $\sum_{t=g}^{g-(\max_{\text{gen}}-1)} \left(\frac{(n_{succ,op}^{t})^{\gamma} + Frac * \sum_{j=1}^{K} n_{succ,j}^{t}}{n_{succ,op}^{t} + n_{fail,op}^{t}} \right) + \epsilon$ (6.13)	$\max_{\text{gen}}$, ϵ γ,$Frac$
Immediate Success $\frac{n_{succ,op}^{g}}{NP}$ (6.14)	-
Fitness Sum	

Success Sum $\frac{\sum_{t=g}^{g-(\max_{\text{gen}}-1)} \sum_{i=1}^{n^t_{succ,op}} OM(t,i,op)}{\sum_{t=g}^{g-(\max_{\text{gen}}-1)} n^t_{succ,op} + n^t_{fail,op}}$	(6.15)	$\max_{\text{gen}}$
Normalised Success Sum Window $\frac{\frac{\sum_{k=1}^{n_{op}} OM(t,k,op)}{n_{op}}}{\left(Best_{j=1}^{K} \frac{\sum_{k=1}^{n_j} OM(t,k,j)}{n_j}\right)^{\omega}}$	(6.16)	ω,$\mathcal{W}$
Normalised Success Sum Generation $\sum_{t=g}^{g-(\max_{\text{gen}}-1)} \frac{\sum_{i=1}^{n^t_{succ,op}} OM(t,i,op)}{n^t_{succ,op} + n^t_{fail,op}}$	(6.17)	$\max_{\text{gen}}$
Best offspring		
Best2Gen $C * \frac{Best_{i=1}^{n^g_{succ,op}} OM(g,i,op) - Best_{i=1}^{n^{g-1}_{succ,op}} OM(g-1,i,op)}{(Best_{i=1}^{n^{g-1}_{succ,op}} OM(g-1,i,op))^{\alpha} * \lvert (n^g_{succ,op} + n^g_{fail,op}) - (n^{g-1}_{succ,op} + n^{g-1}_{fail,op}) \rvert^{\beta}}$	(6.18)	C β, α
Normalised Best Sum $\frac{\frac{1}{\max_{\text{gen}}} \sum_{t=g}^{g-(\max_{\text{gen}}-1)} Best_{i=1}^{n^t_{succ,op}} OM(t,i,op)^{\rho}}{(Best_{j=1}^{K}\{\sum_{t=g}^{g-(\max_{\text{gen}}-1)} Best_{i=1}^{n^t_{succ,j}} OM(t,i,j)\})^{\alpha}}$	(6.19)	α, ρ $\max_{\text{gen}}$

and PR encourage non-dominated operators.

In equation 6.10, proposed in [118], the projection of a coordinate $o_{op}^{fix_{\text{appl}}}$ is taken on a plane represented by an angle θ (a hyper-parameter). This angle defines the trade-off between exploration and exploitation. Thus, the mathematical formulation of projection is given by $|o_{op}^{fix_{\text{appl}}}| \cdot \cos(\alpha_{op})$ where,

$$|o_{op}^{fix_{\text{appl}}}| = \sqrt{(div_{i=k}^{k-(fix_{\text{appl}}-1)} OM(g,i,op))^2 + (qual_{i=k}^{k-(fix_{\text{appl}}-1)} OM(g,i,op))^2} \quad (6.20)$$

$$\alpha_{op} = atan\left(\frac{qual_{i=k}^{k-(fix_{\text{appl}}-1)} OM(g,i,op)}{div_{i=k}^{k-(fix_{\text{appl}}-1)} OM(g,i,op)}\right) - \theta \text{ and} \quad (6.21)$$

α_{op} is the angle between the plane and the coordinate. Compass evaluates the performance of operators by considering not only the fitness improvements from parent to offspring, but also the way they modify the diversity of the population. This was later combined with dynamic multi-armed bandit [120, 117] for adaptive selection of the operators in differential evolution.

Comparison (Rank) The two definitions in this category assign a rank to each operator in the window of size $\mathcal{W}$ according to OM values. These ranks are then decayed using hyper-parameter D to prioritise the operators according to their ranks. As ranking of OM is involved and not the direct OM values, both methods under this category are invariant with respect to the linear scaling of the fitness function. That is, their behavior, when applied on a given fitness function f, is exactly the same when applied to a fitness function defined by $(a \cdot f)$, for any $a > 0$.

Area Under the Curve (AUC) 6.11 [49] plots a Receiver Operator Characteristic (ROC) curve for each operator by scanning the decayed ranking of OM from a window memory. The area is taken as a reward of each operator.

Sum of rank (SR) 6.12 [49] assigns the operators with the sum of the decayed ranks of the OM in the window memory. This sum is normalised by the reward sum of all operators.

Successful operator applications Under this category, the number of successful and unsuccessful applications of each operator are considered. We design two definitions under this category, both using generation memory. For a particular OM under consideration, we count the total number of non-zero values ($n^g_{succ,op}$) as the successful applications of operator op in generation g. Similarly, the number of zeros shows the number of unimproved applications ($n^g_{fail,op}$) of operator op in generation g. These counts are recorded for a fix number of generations represented by hyper-parameter $\max_{\text{gen}}$. Thus there are $\max_{\text{gen}}$ number of values each representing number of successful and unsuccessful applications coming from last $\max_{\text{gen}}$ generations. Mathematically, the number of successes and failures of an operator op in generation g of population size NP is defined as follows:

$$n^g_{succ,op} = \sum_{i=1}^{NP} \begin{cases} 1, & \text{if } OM(g, i, op) > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$n^{g}_{fail,op} = \sum_{i=1}^{NP} \begin{cases} 1, & \text{if } OM(g,i,op) = 0 \\ 0, & \text{otherwise} \end{cases}$$

To calculate the reward of an operator, equation 6.13 [86] takes into account a fraction ($Frac$) of sum of successes of all operators in a generation along with the linear or quadratic contribution of the success of an operator (power is a parameter represented as γ) in the same generation. This term is divided by the total number of applications of the operator in that generation. The total number of applications of an operator in a generation is the total number of successes and failures seen by the operator. This fraction is aggregated for the last max_{gen} generations. In the end, an error value (ϵ) is added that perturbs the resultant reward value. Equation 6.13 is part of the method proposed in [135, 136] and the potential choices of power for γ come from [126, 134]. Another research comes from paper [25] that utilises a similar choice. It proposes MAENSm method that selects a crossover operator among a set of operators. The second term in the numerator of this choice is coming from extension of the ADOPP algorithm in [84].

Next equation 6.14, modified from paper [150], ignores the achievements of the operator in the past. The complete reward definition in the paper can be derived as a combination of 6.14 and 6.27. It is a simple idea defined as the fraction of the immediate or current success with respect to the population size. Here, current success refers to the number of improved OM applications of an operator in the current generation.

Fitness Sum In the previous two categories, the reward definitions have utilised metric values for ranking and calculating successes and failures. In further categories, we will use direct values of metrics to calculate reward.

In fitness sum category, we took three definitions from literature. Equations 6.15 [77] and 6.17 [78] sum the OM values in the last max_{gen} generations from the generation window. The only difference between them is that in the former, once the OM data from all max_{gen} is summed, it is divided by the number of applications in all max_{gen} generations whereas in the latter this division is performed per generation. [165] and [72] considers 6.15 for one generation (that is $\text{max}_{\text{gen}} = 1$). The only difference is that the latter does not include the denominator part.

Equation 6.16 uses the data stored in the window memory. It simply sums the OM values for an operator present in the window of size W divided by its number of applications of the operator present in the window. As the window comprises of successful applications, $n_o p$ denotes the number of applications of op present in the window. This definition for this choice is proposed by [59] known as Average Absolute Reward (AAR). Average Normalised Reward (ANR) proposed in the same research normalises AAR by the best AAR seen by any operator. Thus, we give the normalisation as a choice decided by a hyper-parameter ω.

Best Offspring This category has definitions which consider outliers, that is, it takes into account the OM generated by op that have given best or extreme performance $BestOM(g, i, op)$ in a generation g. This metric value is selected among the successful application from op, $n^g_{succ,op}$. It can easily get trapped in local optima if the landscape is too ragged.

RL-based adaptive methods could not be part of framework fully, as their design is different from AOS in general. Refer to Chapter 7 that presents various deep reinforcement learning models and Chapter 3 for RL literature. RL methods have a concept to reward the operators that produce good offspring. We have included these reward definitions in the framework that are based on the best generated offspring in a generation and Best2Gen 6.18 is the only reward definition inspired from RL design. It is commonly used within the RL design to learn the selection of the operator for each parent. It takes the difference of best seen OM by an operator in the two consecutive generations [113, 78, 21]. The two terms in the denominator, separated by product, are considered by one RL method but discarded by other. Thus, to achieve a general equation we decided to include them with the decision hyper-parameters α and β. [141, 143] consider the best seen OM produced by an operator in the previous generation $g - 1$ along with the difference in the numerator. A hyper-parameter C is multiplied and divided by the difference in applications of operator in the last two generations [100, 99].

Based on the similar idea, equation 6.19 has been used in non-RL context. It uses the best OM value seen in the last $\max_{\text{gen}}$ number of generations generated by an operator [50]. Thus, it not only looks for the best candidate produced by an operator in current and last generation but is far-sighted to combine best fitness in certain

TABLE 6.3: Quality ($q_{g+1,op}$)

Definition	**Parameters**
Weighted Sum $\delta * r_{g+1,op} + (1-\delta) * q_{g,op}$ (6.22)	δ
Upper Confidence Bound $r_{g+1,op} + C \cdot \sqrt{\frac{\log \sum_{j=1}^{K} n_j}{n_{op}}}$ (6.23)	C
Quality Identity $r_{g+1,op}$ (6.24)	-
Weighted Normalised sum $\delta * \max\left\{q_{\min}, \frac{r_{g+1,op}}{\sum_{j=1}^{K} r_{g+1,j}}\right\} + (1-\delta) * q_{g,op}$ (6.25)	$\delta, q_{\min}$
Bellman equation $(1-\gamma P)^{-1} Q'_{g+1}$ (6.26) where, $q'_{g+1,op} = c_1 * r_{g+1,op} + c_2 * r_{g,op}$ (6.27)	c_1, c_2, γ

number of generations. The contribution of best seen value in a generation is either linear or quadratic decided by hyper-parameter ρ in [165]. Optionally, it can be normalised [59] by best *OM* value seen by any operator in last $\max_{\text{gen}}$ generations decided by α. $\alpha = 0, 1$ corresponds to extreme absolute reward and extreme normalised reward respectively. We extend the resultant equation by multiplying it with $\frac{1}{\max_{\text{gen}}}$ [78]. Both equations in this category have a decision parameter α.

6.2.3 Quality

Assigning quality to each operator is an important task involved in AOS method. A quality definition is dependent on current reward and can also include any of the following: reward and quality achieved in previous generation. Thus, to calculate quality, we keep a memory of reward and quality from the previous generation. In the end the quality values are normalised to prevent probability to explode or go out of range. Table 6.3 shows five choices for quality.

Weighted Sum Equation 6.22 in Table 6.3 is the weighted sum of current reward and previous quality. This is part of Probability matching which is originally proposed in [57] and later used as operator selector in AOS [113, 163, 162]. This is the commonly used quality choice in the literature to assign the quality for the adaption of parameters [50, 58, 47].

Weighted Normalised Sum Equation 6.25 has the same definition as equation 6.22 except that the current reward is normalised by the sum of the reward values of all operators to bring the value in the range [0, 1]. This is obtained directly from [78] which involves two hyper-parameters q_{min} and δ. This definition is lower bounded by q_{min}. [77] does not consider a lower bound in the quality, that is $q_{\text{min}} = 0$ given that the reward is positive real value. In case where a reward attained by an operator is zero, the quality becomes a fraction of previous quality. The parameter (δ) in equations 6.22 and 6.25 plays the role to act as a weight for current/normalised reward and previous quality.

Upper confidence Bound (UCB) is a well-known algorithm, originally proposed in [9]. It is known to achieve a compromise between exploitation and exploration. [28] proposes dynamic multi-armed bandit (DMAB), a selection strategy based on UCB. To control the exploration strength, it includes a hyper-parameter C in UCB as shown in equation 6.23. This definition has only been used for window memory where n_j represents the total number of applications of j-th operator present in the window. However, we have made this definition flexible enough such that if a reward choice utilising generation memory is selected, $n^t_{succ,j}$ counts the successful operator applications in generation t by operator j. The formulae calculating UCB value for $\max_{\text{gen}}$ generations is shown below:

$$r_{g+1,op} + C \cdot \sqrt{\frac{\log \sum_{j=1}^{K} \sum_{t=g}^{g-(\max_{\text{gen}}-1)} n^t_{succ,j}}{\sum_{t=g}^{g-(\max_{\text{gen}}-1)} n^t_{succ,op}}} \tag{6.28}$$

In the case of fitness diversity and quality reward definitions, fix_{appl} value for an op is same as n_{op}. This method is also used in [120, 50, 47, 49, 117] either within DMAB or for comparison. UCB variants such as UCB-Tuned [9] and KUCBT [74] are not included as part of the UCB definition to keep the formula simple.

Quality Identity Looking deeply into the AOS methods [135, 134, 26, 15, 126, 118, 117, 84, 86], we identified some existing methods have mixed the definitions of reward and quality. There is no quality component that shares the properties with any of the quality definitions in Table 6.3. Rather these methods could simply be divided into AOS without quality. For instance, SaDE [135] assigns probability to each operator according to the normalised success rate for $\max_{\text{gen}}$ 6.13 and there is no quality definition involved. That is it directly maps success rate reward value of an operator to its probability. Thus, equation 6.24 represents an identity function that maps reward of an operator directly to its quality. In this definition, we do not include previous reward or quality that helps to clearly distinguish the reward from probability definition. It becomes an important choice to determine whether quality has important role in the AOS process.

Bellman Equation Equation 6.26 [150] represents the Bellman equation where each entry in vector Q′ is the weighted sum of the reward values of an operator from previous generations shown in 6.27 [165]. The hyper-parameters c_1 and c_2 denote the weights of these rewards. The adaptive method in [113] considers the sum of c_1 and c_2 to be 1. We lift this condition and each of them can attain a value between 0 and 1. It should be noted that if hyper-parameter $\gamma = 0$ then this definition reduces to just weighted sum of rewards otherwise the Bellman equation is used to calculate the final quality.

6.2.4 Probability

The three probability definitions shown in Table 6.4 are used to assign probability to each operator to get selected in the next generation. These definitions use the current quality of operator and optionally previous probability. Each of the probability choice is lower bounded by a minimum probability of selection $p_{\min}$to avoid probability of any operator becoming zero. An operator showing weak performance in current generation can become useful in later generation. Thus, $p_{\min}$plays important role in allowing an operator to get selected after a certain number of generations. [26, 77, 166] consider generation gap in updating operator probability. That means the probability is updated after certain number of generations. However, in the current framework we have not included this case for simplicity. Thus, we update the probability of each operator at the end of each generation. This helps the method to be

TABLE 6.4: Probability ($p_{g+1,op}$)

Definition		Parameters
Normalised Quality $$p_{\min} + (1 - K * p_{\min}) \left(\frac{q_{g+1,op} + \epsilon_p}{\sum_{j=1}^{K} q_{g+1,j} + \epsilon_p} \right)$$	(6.29)	$p_{\min}, \epsilon_p$
Biased rule $$\begin{cases} \mu * p_{\max} + (1-\mu) * p_{g,op}, & for\ op = max_{j=1}^{K}\{q_{g+1,j}\} \\ \mu * p_{\min} + (1-\mu) * p_{g,j}, & \forall j \neq op \end{cases}$$	(6.30)	μ, $p_{\min}$, $p_{\max}$
Probability Identity $q_{g+1,op}$	(6.31)	-

up-to-date with the operator performance according to the current landscape. We also eliminate the case where each candidate solution in the population is assigned with the probabilities of getting selected by each operator in the population [155]. Instead, we assign probability to each operator and employ a selection mechanism to select an operator for each offspring based on the selection probability of the operator. All the probability definitions are normalised in the end to bring the sum of the probabilities of all operators to 1.

Normalised quality shown in equation 6.29 is the most widely used probability definition in the literature to assign the selection probability to each operator. It is part of Probability matching (PM) originally proposed in [57]. [163, 58, 77, 162, 134, 50, 59, 150, 126, 47, 117, 50, 49, 113, 31, 86, 26] used normalised value of quality, lower bounded by $p_{\min}$. [15, 135, 84, 118, 165] used the normalised value of quality but with $p_{\min} = 0$ and the latter two papers also added an error value (ϵ_p) to this quantity. The error value is used to prevent the quality to become 0. A generalised form of these is shown in 6.29 with two hyper-parameters, $p_{\min}$ and ϵ_p. In the term $(1 - K * p_{\min})$, K denotes the total number of operators. The value for $K * p_{\min}$ should be less than 1 to avoid this term becoming negative. Thus, $p_{\min}$ is dependent on the number of operators employed.

Normalised quality has a disadvantage that it allocates the probabilities to the operators directly proportional to the quality which makes the convergence slow and prevents the exploitation of the operator with maximum probability. To overcome this issue, **biased rule** 6.30 [163] was proposed as part of the Adaptive Pursuit algorithm. It increases the probability of selection of the operator with best quality. This is done by assigning probability to this operator as the weighted sum of the upper bound of probability ($p_{\max}$) and previous probability of the operator. To maintain a large gap between best operator and others, biased rule assigns the latter operators with the weighted sum of $p_{\min}$ and previous probability. It is interesting to note that this definition does not directly include qualities of operators in calculating the probability of an operator. There are three hyper-parameters involved in the biased rule probability definition, namely $p_{\min}$, $p_{\max}$ and μ with $p_{\min} < p_{\max}$. This definition is also utilised in [162, 50].

Probability Identity The last proposed probability definition is shown in equation 6.36. It simply maps the quality of an operator to its probability obtained in the current generation.

6.2.5 Selection

The selection component consists of various choices that are used to select the operator for an individual in the population given the selection probability of each operator. The most commonly used selection method is proportional selection which is popular in Probability Matching technique. Adaptive Pursuit is the only method that utilises greedy selection. In addition to proportional and greedy selection, we present three proposed selection choices based on the combinations of greedy, proportional and linear decay.

Proportional Selection also known as roulette-wheel selection is shown in equation 6.32. The normalised probability of an operator defines its chances of getting selected proportional to its quality in the next generation. That is, the operator with high normalised value has greater chance to get selected compared to the other operators.

Greedy Selection As the name indicates, in 6.33 the operator with the maximum probability is selected in the next generation. This choice is known to bring least exploration in the task.

TABLE 6.5: Selection(op)

Name	Definition	Parameter
Proportional	$\frac{p_{g+1,op}}{\sum_{j=1}^{K} p_{g+1,j}}$ (6.32)	-
Greedy	$\max_{j=1}^{K}\{p_{g+1,j}\}$ (6.33)	-
Epsilon-Greedy	$\begin{cases} rand[1,K], & if\ uniform(0,1) < eps \\ \max_{j=1}^{K}\{p_{g+1,j}\}, & else \end{cases}$ (6.34)	eps
Linear-Annealed	$\begin{cases} rand[1,K], & if\ uniform(0,1) < Anneal_{eps}(0,1) \\ \max_{j=1}^{K}\{p_{g+1,j}\}, & else \end{cases}$ (6.35)	-
Proportional-Greedy	$\begin{cases} \frac{p_{g+1,op}}{\sum_{j=1}^{K} p_{g+1,j}}, & if\ uniform(0,1) < eps \\ \max_{j=1}^{K}\{p_{g+1,j}\}, & else \end{cases}$ (6.36)	eps

Epsilon-Greedy Selection Greedy selection has a drawback that it neglects the exploration aspect which plays an important role in searching the unexplored parts of the search space. Thus, we introduced a novel selection choice named epsilon greedy selection 6.34. The hyper-parameter eps ensures that a random strategy (rand[1,K]) is selected (exploration). $uniform(0,1)$ indicates a uniformly selected number between 0 and 1.

Linear-Annealed Selection The epsilon-greedy definition in 6.35 keeps the eps value static during the whole run of an EA. However, adapting eps can ensure a right balance between exploration and exploitation. This is so because at different stages of algorithm, importance of exploration and exploitation varies. The higher the value of eps, the greater the exploration will be. Thus, we propose a linear decay of eps where vaue of eps decreases from 1 to 0 as the algorithm progresses represented by $Annealed_{eps}(0,1)$. Thus, this ensures that algorithm explores in the initial runs and exploits towards the end.

Proportional-Greedy Selection This selection choice 6.36 is a hybrid of proportional and greedy selection. If a random number between 0 and 1 is smaller than the hyper-parameter, ϵ, proportional selection is performed otherwise greedy is performed. Here the random operator is not selected for exploration but the one that has shown good performance (and not necessarily the best) recently. This brings a restricted exploration of the search space compared to the equation 6.34.

6.3 AOS methods utilised to build the framework

In the previous section we presented the proposed framework utilising various AOS methods from literature. The component choices sharing same properties are generalised with the introduction of a number of hyper-parameters. It is possible to replicate many AOS methods existing in literature using the designed framework. A particular AOS method is obtained by setting the component choices along with their hyper-parameter values. The Table 6.6 shows the AOS methods from the literature that can be replicated from the unified framework. The table shows a method as a combination of component choices from the framework with their hyper-parameter values as proposed in the literature. The first column gives the method name and the rest of the columns indicate a choice from each component setting their hyper-parameter values as decided in their respective papers. Not all algorithms shown in the table are given a name, thus we decide to name these algorithms such that it best suits the description of the algorithm.

[165] proposes an AOS method, named as Dyn-GEP, in the context of Gene Expression programming (GEP). It assigns probabilities to the operators as follows:

$$p_i(t) = \frac{d_i(t)}{\sum_{i=1}^{n} d_i(t)} \text{ and } d_i(t) = d_0 + m_i(t) + \alpha * m_i(t-1) \tag{6.37}$$

where, $p_i(t)$, $d_i(t)$ and $m_i(t)$ is the probability, improvement and mean value of reward assigned to an operator i at generation t respectively; α represents the forgetting factor; n is the number of operators and d_0 is the minimum value attained by $d_i(t)$. We simplify this formula to map to each component. It assigns fitness improvement w.r.t. parent 6.2 as offspring metric. This method runs the algorithm 20 times and considers two rewards, best fitness value in 20 runs and mean of best value found in each 20 runs. As per the design of the framework, instead of running the algorithm multiple times, we run the method only once. Thus, we assign reward as mean of fitness values 6.15 produced by the application of the operator in the current

TABLE 6.6: Relevant literature

AOS Methods	***OM***	**Reward**	**Quality**	**Probability**	**Selection**
Hybrid	6.1	6.18 ($\alpha = \beta = 0, C = 1$)	6.26 ($c_1 + c_2 = 1, \gamma = 0$)	6.29 ($p_{\min} = \epsilon_p = 0$)	6.32
Op-adapt	6.3	6.17	6.25	6.31	6.32
PDP	6.2	6.13 ($\gamma = 2$, $\max_{\text{gen}} = 1$, $Frac = \epsilon = 0$)	6.24	6.29 ($\epsilon_p = 0, p_{min} = \lfloor \frac{20}{K} \rfloor$)	6.32
ADOPP	6.5	6.13 ($\epsilon = 0$, $\gamma = 1, \max_{\text{gen}} = 1$, $Frac = 0$)	6.24	6.29 ($p_{\min} = \epsilon_p = 0$)	6.32
ADOPP-ext	6.5	6.13 ($\epsilon = 0, \gamma = 1$, $\max_{\text{gen}} = 1$)	6.24	6.29 ($p_{\min} = \epsilon_p = 0$)	6.32
Adapt-NN	6.2	6.15	6.25 ($q_{\min} = 0$)	6.29 ($\epsilon_p = 0$)	6.32
Dyn-GEPv1	6.2	6.15 ($\max_{\text{gen}} = 1$)	6.26 ($c_1 = 1, \gamma = 0$)	6.29 ($p_{\min} = 0$)	6.32
Dyn-GEPv2	6.2	6.19 ($\rho = 3$, $\alpha = 0, \max_{\text{gen}} = 1$)	6.26 ($c_1 = 1, \gamma = 0$)	6.29 ($p_{\min} = 0$)	6.32
SaDE	6.2	6.13 ($\gamma = 1, Frac = 0$)	6.24	6.29 ($p_{\min} = 0, \epsilon_p = 0$)	6.32
MMRDE	6.2	6.13 ($\max_{\text{gen}} = \gamma = 1, Frac = \epsilon = 0$)	6.24	6.29 ($\epsilon_p = 0$)	6.32
Compass	6.2	6.10	6.24	6.29 ($p_{\min} = 0$)	6.32
PD-PM	6.2	6.8	6.22	6.29 ($\epsilon_p = 0$)	6.32
PR-PM	6.2	6.9	6.22	6.29 ($\epsilon_p = 0$)	6.32
Proj-PM	6.2	6.10	6.22	6.29 ($\epsilon_p = 0$)	6.32
F-AUC-MAB	6.2	6.11	6.23	6.29 ($\epsilon_p = p_{\min} = 0$)	6.33
F-SR-MAB	6.2	6.12	6.23	6.29 ($\epsilon_p = p_{\min} = 0$)	6.33
F-AUC-AP	6.2	6.11	6.22	6.30	6.32

F-SR-AP	6.2	6.12	6.22	6.30	6.32
F-AUC-AP	6.2	6.11	6.22	6.29	6.32
F-SR-PM	6.2	6.12	6.22	6.29	6.32
RecPM	6.2	6.14	6.26 ($c1 = 1$, $c2 = 0.5$, $\gamma = 0.46$)	6.29 ($\epsilon_p = 0, p_{\min} = 0.11$)	6.32
MAENSm	6.1	6.13 ($\max_{\text{gen}} = \gamma = 1, Frac = \epsilon = 0$)	6.23	6.29	6.32
PM-AdapSS-AA	6.6	6.16 ($\omega = 0$)	6.22	6.29 ($\epsilon_p = 0$)	6.32
PM-AdapSS-N	6.6	6.16 ($\omega = 1$)	6.22	6.29 ($\epsilon_p = 0$)	6.32
PM-AdapSS-EA	6.6	6.19 ($\alpha = 0, \rho = 1$)	6.22	6.29 ($\epsilon_p = 0$)	6.32
PM-AdapSS-N	6.6	6.19 ($\alpha = 1, \rho = 1$)	6.22	6.29 ($\epsilon_p = 0$)	6.32
Ex-PM	6.2	6.19 ($\rho = 1, \alpha = 0$)	6.22	6.29	6.32
Ex-AP	6.2	6.19 ($\rho = 1, \alpha = 0$)	6.22	6.30	6.32
Ex-MAB	6.2	6.19 ($\rho = 1, \alpha = 0$)	6.23	6.29 ($\epsilon_p = p_{\min} = 0$)	6.33

generation (not run) and best operator application in the current generation 6.19. We consider Dyn-GEP with two versions named Dyn-GEPv1 and Dyn-GEPv2 for two reward definitions, respectively. Dyn-GEPv1 calculates mean fitness value from current generation ($\max_{\text{gen}} = 1$) where the sum in the denominator of equation 6.15 is the total number of application of an operator. Equation 6.19 in Dyn-GEPv2 sets $\max_{\text{gen}}$ as 1 with $\rho = 3$ without normalisation ($\alpha = 0$). These two definitions represent $m_i(t)$ in equation 6.37. The quality can be extracted from the above equation to map to equation 6.26 where hyper-parameters γ and c_1 are set as 0 and 1 respectively and c_2 represents α in the above equation. Moving to probability, it is represented by equation 6.29 with ϵ_p set as d_0 and $p_{\min} = 0$. Proportional selection is used to select operators based on the probability.

Hybrid algorithm, presented in [113], is an algorithm that picks among two elitist algorithms. This approach is no different than AOS methods that are used to select operators. Thus, we decided to include it in the framework as a method to select among discrete choices. It uses raw fitness values 6.1 as offspring metric and their difference in two consecutive generations to assign reward to each choice. As soon as one of the algorithms is applied to evolve two individuals from the population, reward is assigned as a choice in reward component, Best2Gen 6.18 is utilised with the hyper-parameters $C = 1$, α and $\beta = 0$. It then combines current and previous reward 6.26 of an algorithm as weighted geometric average with coefficient c_1 and c_2 summing to 1. It does not consider future reward, so γ is taken as 0. Finally, probability 6.29 and selection 6.32 definitions according to the probability matching is used as an assignment rule to probability and selection of an algorithm for next evolution.

ADOPP [86] is one of the early AOS methods proposed in 1995. It assigns each operator a probability proportional to the contribution it has made in producing an offspring better than population median. This operator receives a reward of 1.0 and assigns a decayed reward to its ancestors. However, we do not consider the possibility of rewarding the ancestors. To avoid the probability becoming zero, an extension of ADOPP is proposed, ADOPP-ext [84]. In ADOPP-ext, a fraction of the sum of all operators' reward is added to each operator's accumulated reward. Population-level Dynamic probabilities (PDP) [126] is a similar approach to ADOPP where instead of offspring comparison to median population fitness, it considers operation applications that improved fitness w.r.t parent. Other component choices in PDP are matched with ADOPP with the variation in their hyper-parameter values.

The SaDE algorithm [135] combines adaptive control of the mutation and crossover rate with adaptive selection of two mutation operators. As our proposed framework is concerned with unifying AOS methods, we only show to replicate the AOS method involved in SaDE from the framework. Operators are assigned proportional to the probability of operator selection. The probability is based on the success and failure rate in previous fix number of generation. SaDE and MMRDE [134] share the same component choices. The only difference is that latter assigns reward based on the applications in certain number of generations whereas the former only considers the applications in current generation.

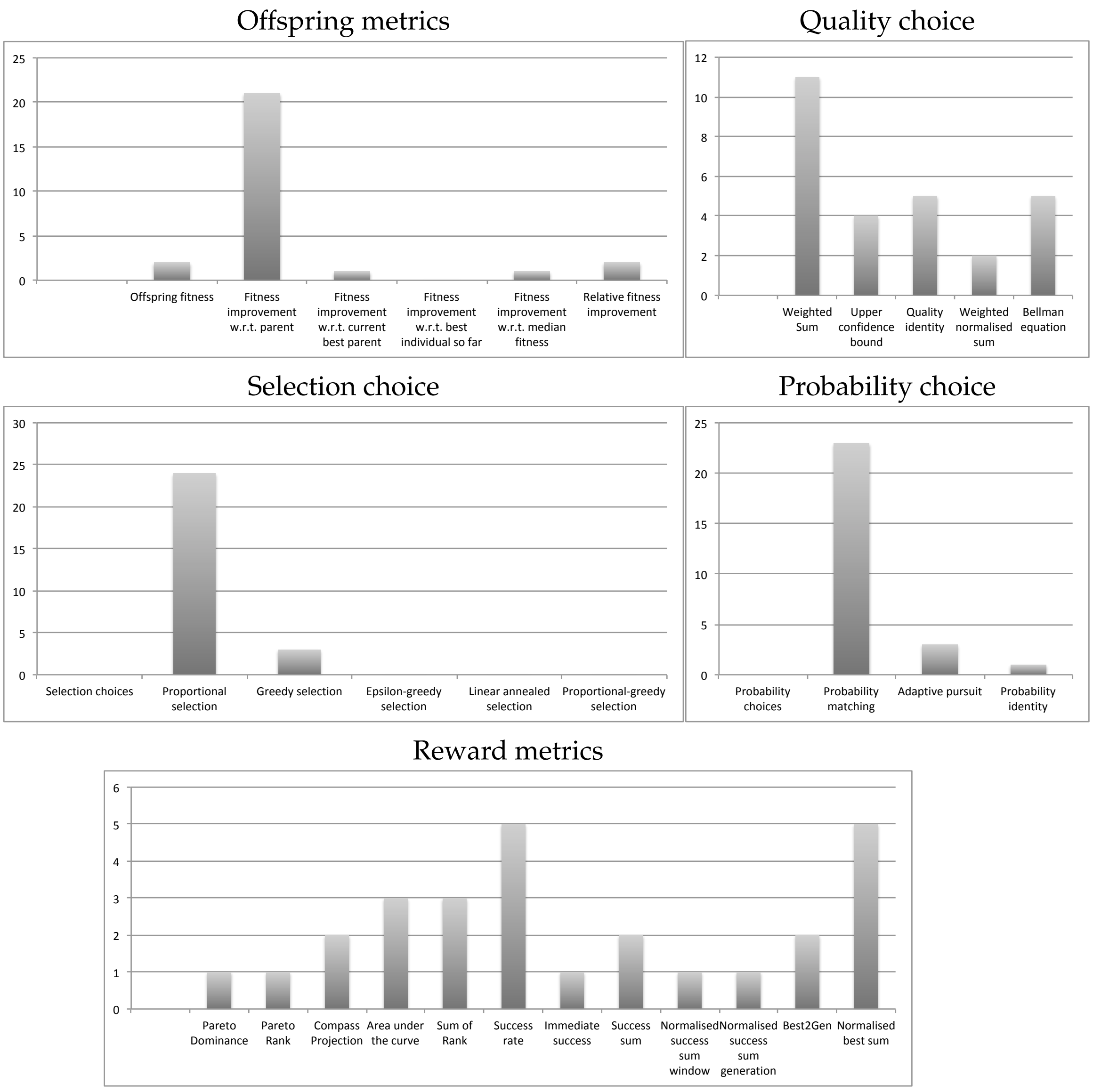
RecPM-AOS [150] is a recently proposed AOS method that has shown promising results explained in detail in the previous Chapter 5. It is based on the idea of maximising the future reward utilising probability matching mechanism. *Compass*, *F-AUC-MAB* and *PM-AdapSS* are among popular AOS methods that can be replicated by fixing choices for each component in the framework. An overview of these methods can be found in Chapter 3.

The usage of different component choices involved in adaptive methods vary greatly in the literature. We attempt to analyse the trend of utilising different choices in each component extracted from Table 6.6. The summary of choice count considered in literature is shown in Figure 6.4. It can be clearly seen that a lot of emphasis is given on exploring novel reward choices among all components while proposing an AOS method. Success rate (6.13) and normalised best sum (6.19) are among the popular choices for reward. In other components, Fitness improvement w.r.t. parent (6.2) is frequently used offspring metric and probability matching is a common choice by researchers that assign quality (6.22), probability (6.29) and selection mechanism (6.32). The combinations of these popular component choices have been explored in literature. [117] considers three reward definitions based on fitness improvement w.r.t. parent namely, pareto rank, pareto dominance and projection. To form a complete AOS method, these are combined with PM. Six combinations are tested in [49], area under the curve and sum of rank reward choices paired with dynamic multi-armed bandit (6.23 + 6.29 + 6.33), adaptive pursuit (6.22 + 6.30 + 6.32) and probability matching. We have not included statistical factor in the framework, and have only considered MAB instead of D-MAB. The papers testing other combinations include [50, 59, 119].

Besides these combinations, many other novel AOS methods can be designed using the framework. These combinations can be easily tested with differential evolution using the tool available on Github [1]. The framework can be utilised to control the discrete parameters of not only DE but also any other evolutionary algorithm that has discrete parameters.

[1]https://github.com/mudita11/Tune-AOS-bbob

FIGURE 6.4: Component choices with their usage frequency in the literature



6.4 Experimental Design

As seen in the previous chapter *JaDE* showed competitive results compared to the other algorithms. Thus, we decided to include the mutation strategy involved in *JaDE* within the list of operators to be adapted by an AOS method. We also include three other popular mutation strategies to increase the robustness of the framework. In addition to the strategies adapted in the previous chapter, four other strategies included are “curr-to-pbest/1(archived)”, “current-to-best/1”, “best/1”, “best/2”and “curr-to-pbest/1”. Thus, in total nine strategies are adapted within DE, shown in section 2.2.3.

6.4.1 Parameter tuning

The framework consists of $6 \times 12 \times 5 \times 3 \times 5 = 5,400$ AOS methods. These methods have their own hyper-parameters which increase the number of unique combinations possible. As the parameter space is huge, we decided to combine the framework with an offline method that tunes the framework and returns a combination of choices for a problem set. We use the offline automatic configurator *irace* to tune the framework. An overview of *irace* can be found in Chapter 4. It is provided with a training set and component choices along with their hyper-parameters. In addition to tuning the framework and its hyper-parameters, it also tunes the parameters of the DE algorithm F, NP, CR and top_{NP}. Parameter ranges/choices given to *irace* are shown in Table 6.7. The table shows the name, type (Real, Integer, categorical) and range of each parameter tuned by *irace*. The real valued parameters take a two-point float value, integer type represents integer value in the given range and categorical can only select from the choices given. The budget given to *irace* is 10^4. The candidate configuration sampled by *irace* is an AOS method with a set of its hyper-parameter values and tuned DE parameters. The budget assigned to a run is $10^4 \cdot n$ function evaluations where $n = 20$ is the dimension of the function. In this chapter, all algorithms focus on $n = 20$ for all functions.

The training set consists of 48 function instances of the BBOB noiseless functions of dimension 20. Details on the BBOB benchmark set can be found in Chapter 4. We give four AOS methods as the starting configurations to *irace*. *Compass* [118],

TABLE 6.7: Hyper-parameter choices given to *irace*

Parameter Name	**Type**	**Range**	**Notes**
		DE parameters	
F	Real	[0.1, 2.0]	Mutation Rate
CR	Real	[0.1, 1.0]	Crossover Rate
NP	Integer	[50, 400]	Population Size
top_{NP}	Real	[0.02, 1.0]	Top p candidates
		Component choices	
Offspring Metric	Categorical	[0, 6]	Type of *OM*
Reward Type	Categorical	[0, 11]	Type of Reward
Quality Type	Categorical	[0, 4]	Type of Quality
Probability Type	Categorical	[0, 2]	Type of Probability
Selection Type	Categorical	[0, 4]	Type of Selection
		Reward Choice parameters	
fix_{appl}	Integer	[10, 50]	Fix number of applications
$\max_{\text{gen}}$	Integer	[1, 50]	Maximum number of generations
θ	Categorical	(36, 45, 54, 90)	Projection angle
$\mathcal{W}$	Integer	(20, 150)	Size of window
D	Real	[0.0, 1.0]	Decay factor
γ	Categorical	(1, 2)	Success choice
$Frac$	Real	[0.0, 1.0]	Fraction of overall success
ϵ	Real	[0.0, 1.0]	Noise
ω	Categorical	(0, 1)	Normalisation choice
C	Real	[0.001, 1.0]	Scaling constant
α	Categorical	(0, 1)	Decision parameter
β	Categorical	(0, 1)	Decision parameter
ρ	Categorical	(1, 2, 3)	Intensity
		Quality Choice parameters	
δ	Real	(0.0, 1.0)	Decay rate
C	Real	(0.0, 1.0)	Scaling Factor
q_{min}	Real	(0.01, 1.0)	Minimum quality attained
c_1	Real	(0.0, 1.0)	Memory for current reward
c_2	Real	(0.0, 1.0)	Memory for previous reward
γ	Real	(0.01, 1.0)	Discount rate
		Probability Choice parameters	
$p_{\min}$	Real	[0.0, 1.0]	Minimum selection probability
ϵ_p	Real	(0.0, 1.0)	Noise
μ	Real	(0.0, 1.0)	Learning rate
$p_{\max}$	Real	(0.0, 1.0)	Maximum selection probability
		Selection Choice parameters	
eps	Real	[0.0, 1.0]	Random probability of selection

TABLE 6.8: Starting configurations and the configuration returned by *irace*

Parameter name	***RecPM-AOS***	***PM-AdapSS***	***F-AUC-MAB***	***Compass***	**Configuration returned by *irace* (*U-AOS-FW*)**
DE parameters					
F	0.57	0.47	0.45	0.51	0.41
CR	0.93	0.96	0.21	0.95	0.91
NP	154	329	57	163	262
p in pbest	0.05	0.07	0.73	0.64	0.02
Component choices					
Offspring Metric	6.2	6.2	6.2	6.2	6.2
Reward Type	6.14	6.16	6.11	6.10	6.14
Quality Type	6.26	6.22	6.23	6.24	6.26
Probability Type	6.29	6.29	6.29	6.29	6.29
Selection Type	6.32	6.32	6.33	6.32	6.32
Reward Choice hyper-parameters					
fix_{appl}	-	-	-	66	-
θ	-	-	-	90	-
$\mathcal{W}$	-	73	138	-	-
D	-	-	0.47	-	-
ω	-	1	-	-	-
Quality Choice hyper-parameters					
δ	-	0.07	-	-	-
c	-	-	0.04	-	-
c_1	0.57	-	-	-	0.66
c_2	0.96	-	-	-	0.45
γ	0.43	-	-	-	0.54
Probability Choice hyper-parameters					
p_{min}	0.08	0.06	0.02	0.08	0.04
ϵ_p	0.26	0.53	0.72	0.55	0.22

PM-AdapSS [59] and *F-AUC-MAB* [49] are among popular AOS methods along with *RecPM-AOS* that act as initial population for *irace* to explore the parameter search space. We tune the hyper-parameters of the starting configurations along with DE parameters using *irace*. The tuned AOS methods within DE are shown in Table 6.8.

6.5 Testing phase

After tuning, *irace* returns an AOS method along with its tuned hyper-parameter values and tuned DE parameter values given the starting configurations. The last column in 6.8 shows the configuration returned by *irace*, abbreviated as *U-AOS-FW*. The returned configuration is a variant of *RecPM-AOS* described in previous Chapter 5.

RecPM-AOS assigns reward to an operator depending on the short term success of that operator and estimates quality based on the expected quality of possible selection of operators in the past. It shares similarities with *PM-AdapSS* AOS method. When comparing *PM-AdapSS* and *RecPM-AOS*, the former uses *PM* to select an operator whereas the latter uses a variant of *PM* known as *RecPM*. Both AOS methods use reward based on the number of improvements from parent to offspring, however, *PM-AdapSS* uses average relative fitness improvement as immediate reward without using accumulated reward, whereas *RecPM-AOS* uses offspring survival rate as immediate reward combined with a fraction of its previous accumulated reward.

Algorithm 5 shows the working steps of AOS within DE in the testing phase. For a given test problem, this is simply the working of DE with multiple mutation operators where each parent is evolved with an operator using the selection method employed in the AOS method. Testing starts by initialising and evaluating the parent population. The *OM* values are calculated for each offspring to initialise the generation and window memory. The probability for each operator is initialised as $\frac{1}{K}$ where K is the total number of mutation strategies. This gives every operator an equal chance to get selected. Once the initialisation phase is over, the following steps are repeated as long as the stopping criteria are not satisfied. The parent population is evolved using a mutation strategy selected for each parent by selection definition in the AOS method. The selection is performed based on the probability of each operator. The memory is updated based on *OM*. The offspring population is evaluated and the solution among parent and offspring with better fitness survives. This is followed by reward, quality and probability update according to the AOS method. Once the algorithm terminates, best fitness value is returned.

We select four tuned AOS methods within DE *PM-AdapSS* [59], *F-AUC-MAB* [49], *Compass* [118] and *RecPM-AOS* [150]; two non-AOS DE algorithms with one mutation strategy, *JaDE* [169] and *R-SHADE* [160] to compare with the returned configuration. *PM-AdapSS* and *F-AUC-MAB* were introduced in the context of selecting a mutation strategy aka operator in (DE) [132]. In particular, *PM-AdapSS* uses probability matching as the method for operator selection, whereas *F-AUC-MAB* employs

Algorithm 5 AOS method formed with the component choices from the framework coupled with DE

```
1: Given: Tuned $CR$, $F$ and $NP$; Component choices from AOS framework
2: Initialise and evaluate fitness of each individual $x_i$ in the population
3: $g = 0$ (generation number)
4: Calculate $OM(0, k, op)$, $\forall$ k
5: Initialise generation and window memory using $OM(0, k, op)$, $\forall$ k
6: Initialise probability $P_{0,op}$, $\forall op \in Op$
7: while stopping condition is not satisfied do
8:    for each $x_i$, $i = 1, \ldots, NP$ do
9:       if one or more operators not yet applied then
10:         $op$ = Uniform selection among operator(s) not yet applied
11:      else
12:         $op$ = Select a mutation strategy based on the selection choice
13:      Generate offspring using selected operator $op$
14:   Calculate $OM(g, k, op)$
15:   Update generation and window memory using $OM(g, k, op)$, $\forall$ k
16:   Evaluate offspring population
17:   Perform survival selection
18:   Calculate reward for each operator $R_{g+1,op}$
19:   Estimate quality for each operator $Q_{g+1,op}$
20:   Update probability for each operator $P_{g+1,op}$
21:   $g = g + 1$
```

a method inspired by multi-armed bandits. *Compass* evaluates an operator's impact using two measures, mean fitness and population fitness diversity. It measures how well the operator balances the exploration and exploitation. The latter two algorithms employ different ways to self-adapt crossover and mutation rate in DE. *JaDE* is a DE variant that adapts the crossover probability CR and mutation factor F using the values which proved to be useful in recent generations. *R-SHADE* is an improvement upon *JaDE* which employs a restart mechanism and uses a parameter adaptation mechanism based on a historical record of successful parameter settings to adapt CR and F. *JaDE* and *R-SHADE* utilise same mutation strategy "current-to-pbest" to evolve population and it is one of the operators adapted by returned configuration from unified framework, *U-AOS-FW* and other AOS methods in comparison.

The data for non-AOS methods is taken from the COCO website.[2] As we clearly

[2]http://coco.gforge.inria.fr/doku.php?id=algorithms-bbob

outperform *CMA-ES* in the previous chapter, we decided not to compare the returned configuration with *CMA-ES*. The returned configuration and other six algorithms from the literature are evaluated on the remaining 312 BBOB noiseless benchmark set on dimension 20. The budget allocated to a run is $10^5 \cdot 20$ function evaluations. We use plots of the Empirical Cumulative Distribution Function (ECDF) to assess their performance (Fig. 6.6). The introduction on ECDF graphs can be found in section 5.2.1 of Chapter 5. The aRT tables are shown in Appendix C.

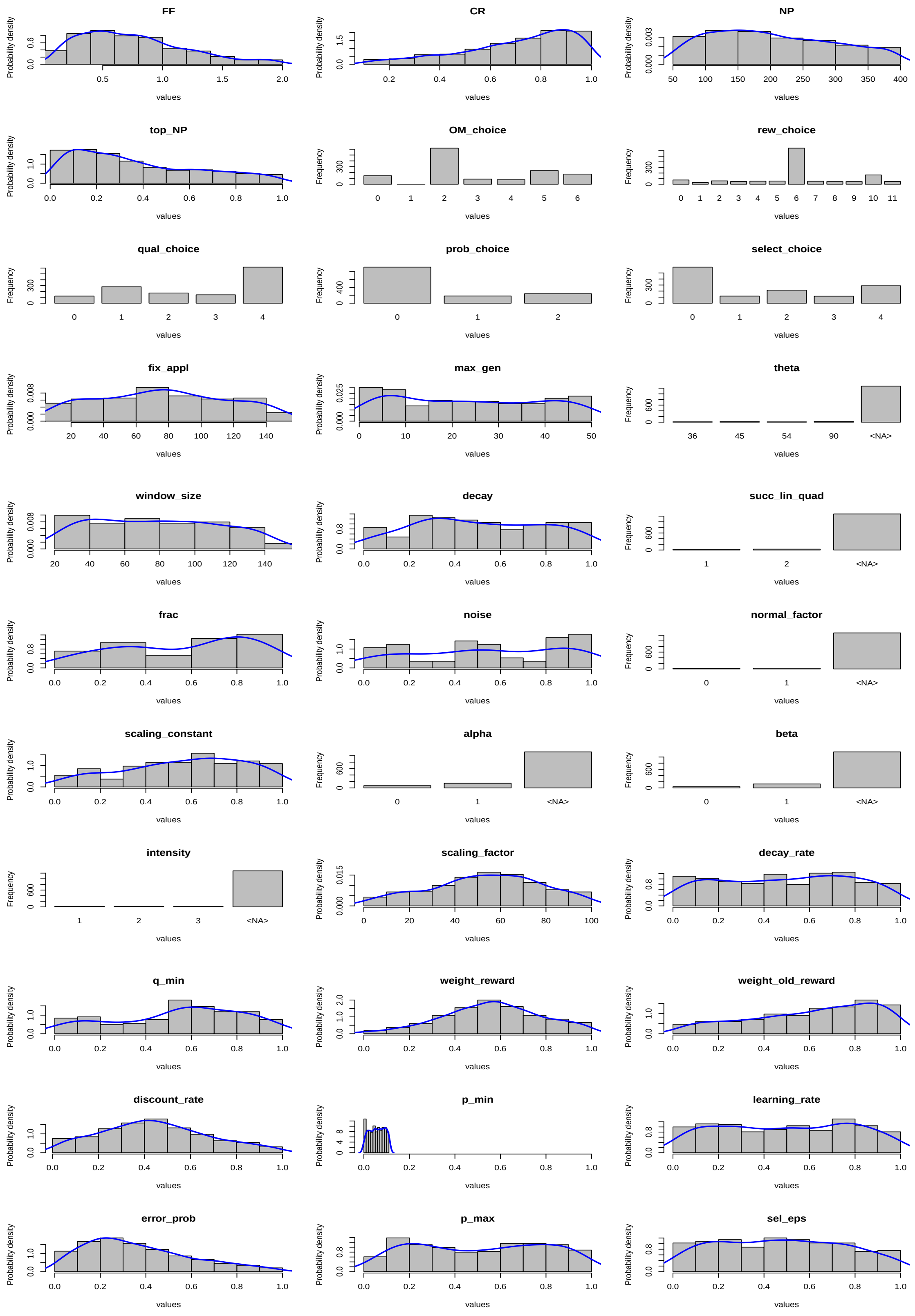
6.6 Experiments and Results

During tuning *irace* samples parameters from their individual parameter distribution. The sampling frequency of the parameters is shown in Figure 6.5. Categorical parameters that take values from a set of values are shown in the figure such as OM_choice and frac. Parameters that take floating-values such as FF and decay sample from a probability distribution such as a normal distribution.

6.6.1 Comparison of *U-AOS-FW* with other tuned AOS methods

ECDF graphs on 24 functions each with 13 test instances are shown in Figure 6.6. It also shows class-wise and overall performance on the test set. These graphs clearly show that the configuration returned by *irace*, *U-AOS-FW* reaches all targets with competitive speed for functions f001, f002 and f005-f014 in comparison with *RecPM-AOS*, *PM-AdapSS*, *Compass* and *F-AUC-MAB*. Looking at the separable class *Compass* and *U-AOS-FW* solves problems with the same speed solving the same number of problems as *PM-AdapSS*. Although *U-AOS-FW* is a variant of *RecPM-AOS*, the former solves more problems than the latter with faster speed. *F-AUC-MAB* has shown exceptional performance in terms of speed and problems solves compared to other AOS methods for separable class of problems. *U-AOS-FW* performed best for low/moderate conditioning problems reaching all targets for all 4×13 test instances falling under this category and the same performance of all AOS methods can be seen in the case of high conditioning class of problems. *F-AUC-MAB* performed worse on low/moderate and high conditioning class, solving the least number of problems. *U-AOS-FW* and *RecPM-AOS* excelled in the multi modal (adequate

FIGURE 6.5: Parameter sampling frequency. FF for F, top_NP for p in pbest, θ for theta, $\mathcal{W}$ for window, D for decay, γ for succ_lin_quad, Frac for frac, ϵ for noise, ω for normal_factor, C for scaling_constant, α for alpha, β for beta, ρ for intensity, c for scaling_factor, δ for decay_rate, c1 for weight_reward, c2 for weight_old_reward, γ for discount_rate, μ for learning_rate, ϵ_p for error_prob, eps for sel_eps



structure) class of problems. They showed similar performance with former solving faster than any other algorithm. *U-AOS-FW* came second in the multi modal (weak structure) problems after *F-AUC-MAB* which solved more than 30% problems. Overall, *U-AOS-FW* solved approximately 65% of total problems which is more than the number of problems solved by any tuned AOS method and also with faster speed compared to any AOS method.

6.6.2 Comparison of *U-AOS-FW* with non-AOS methods

We compare the performance of *U-AOS-FW* with non-AOS methods that is *CMA-ES*, *JaDE* and *R-SHADE*. The latter two are variants of DE utilising one strategy to evolve population of candidate solutions. CMA-ES is a variant of evolutionary strategy. *U-AOS-FW* outperformed the three algorithms in three classes namely multi modal (adequate structure), low/moderate and high conditioning problems. It excelled not only in solving maximum number of problems in these three class but also with higher speed. Overall, *U-AOS-FW* solved more problems than CMA-ES and ranked third after *JaDE* and *R-SHADE*.

6.6.3 Comparison of *U-AOS-FW* trained on nine operators with *RecPM-AOS* on four operators

We are interested in analysing whether adding new operators in the set of operators for AOS method to choose from lead to any improvement in performance. Thus, we made an attempt in analysing the difference in performance of *U-AOS-FW* trained on nine operators to *RecPM-AOS* trained on four operators (*RecPM1*). To analyse this, we compare the class-wise ECDF graphs of *U-AOS-FW* shown in Figure 6.6 and *RecPM-AOS* shown in Figure 5.3 from Chapter 5. There does not seem be any difference in performance of *U-AOS-FW* and *RecPM1* in solving separable, multi modal (adequate), low and high conditioning class problems, that is they solve the same number of problems. However, *U-AOS-FW* solves the function instances of low/moderate conditioning and multi modal (adequate structure) with greater speed. In solving multi modal (weak structure), *U-AOS-FW* outperforms RecPM1

FIGURE 6.6: ECDFs on test set. F-AUC for *F-AUC-MAB*, UFW for *U-AOS-FW*, RecPM for *RecPM-AOS*, AdapSS for *PM-AdapSS*

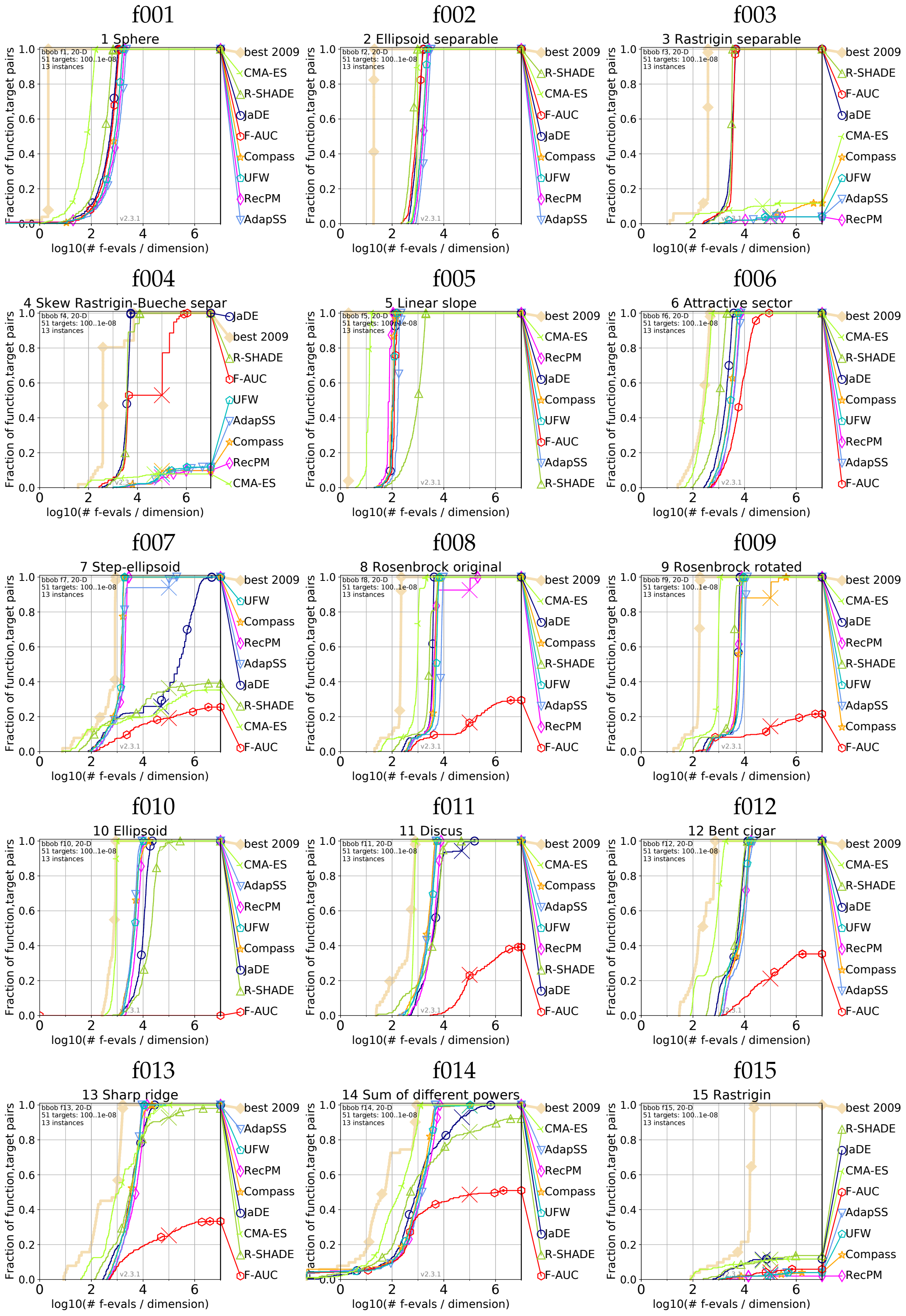
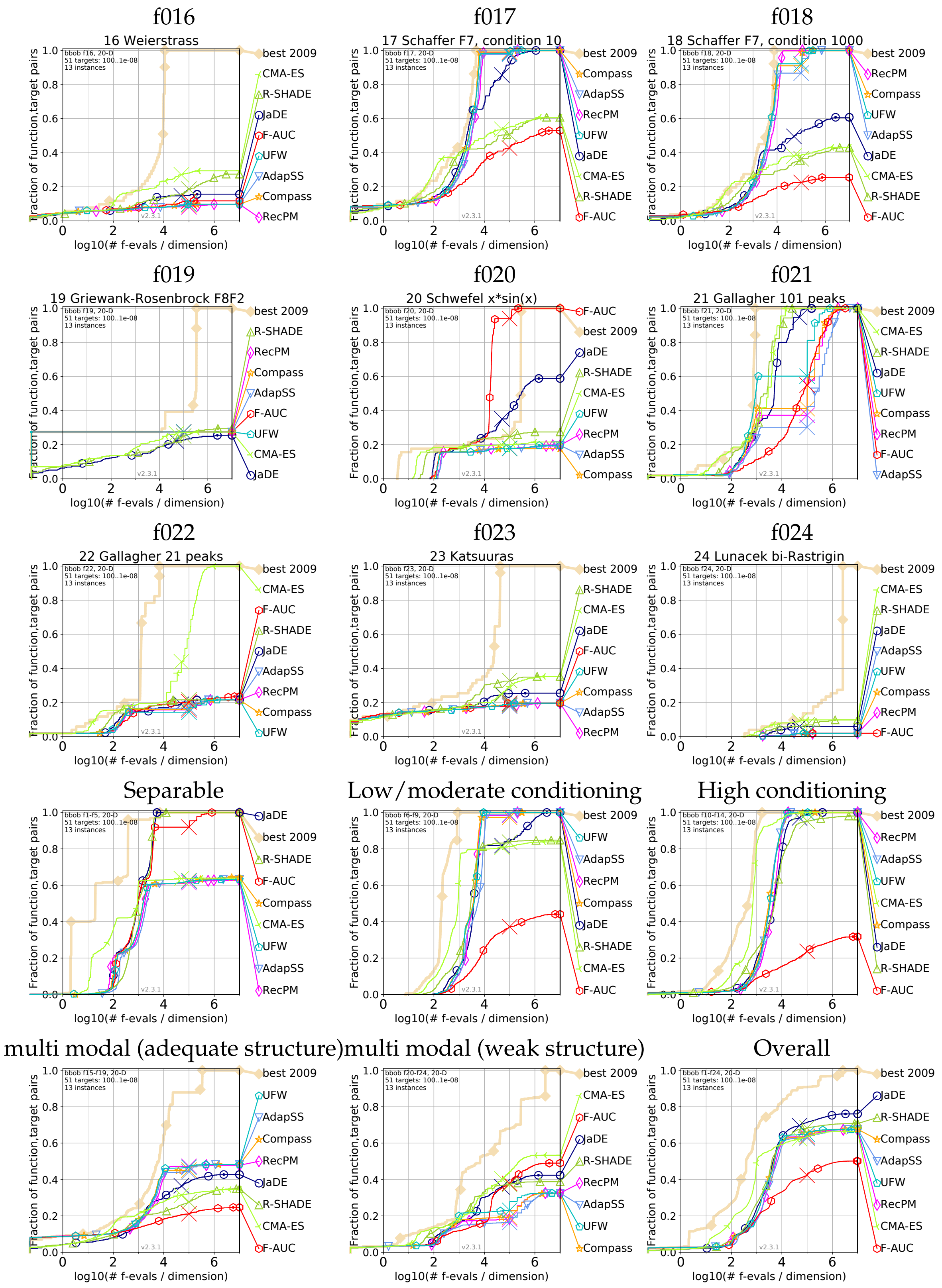


FIGURE 6.6 (cont.): ECDFs on test set. F-AUC for *F-AUC-MAB*, UFW for *U-AOS-FW*, RecPM for *RecPM-AOS*, AdapSS for *PM-AdapSS*



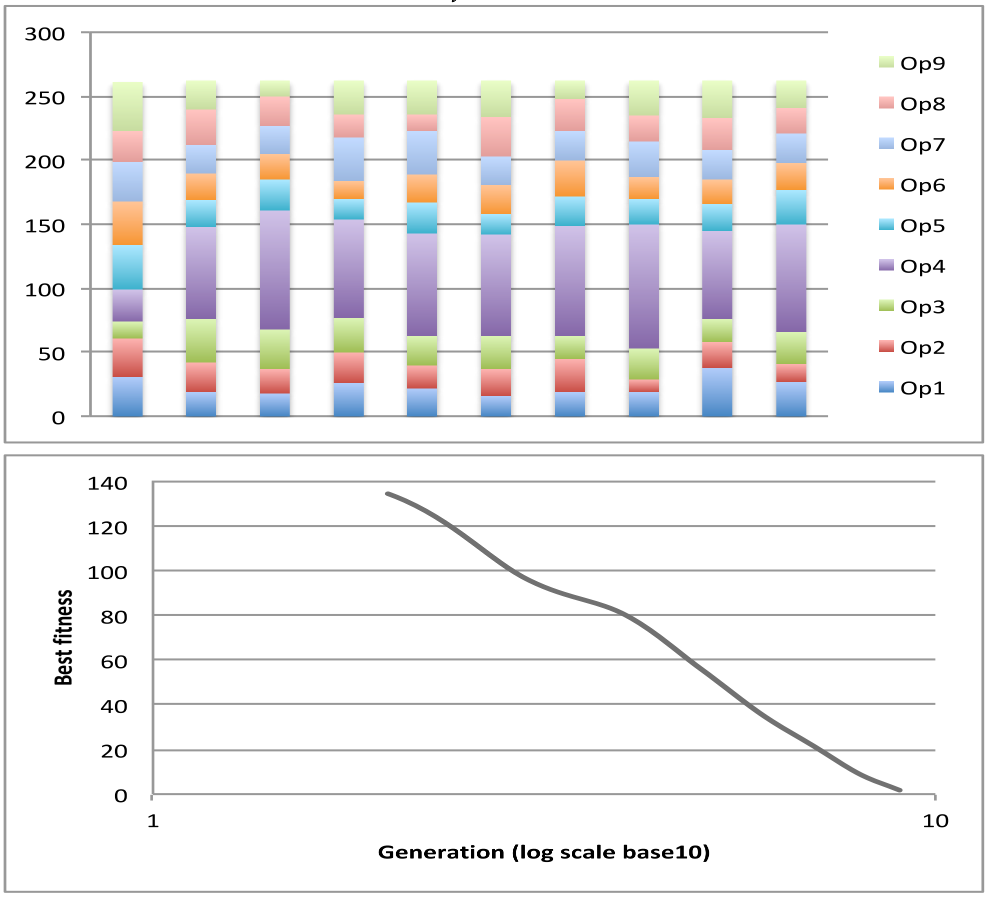
in terms of speed and different targets. Thus, adding more operators helped solving more problems with higher speed, showing a better arrangement for exploration and exploitation of search space.

Next we try to identify the operators that lead to differences in performance of these two algorithms. Thus, further to analysing ECDF graphs we plan to better understand the impact of the selected operators by comparing the graphs generated as a result of running algorithms. We compare the performance of *RecPM1* (Figure 5.4) and by *U-AOS-FW* (Figure 6.7) on functions $f05$ and $f07$ each with instance $i01$. Figure 5.4 is a result of four operator applications by *RecPM1* and Figure 6.7 is a result of utilising nine different operators by *U-AOS-FW* in different generations. A value of $p_{\text{min}} = 0.04$ controls the level of adaptation of operators. To distinguish these operators, we represent them with different colors. The operators utilised by different candidate solutions to produce offspring in a generation are represented by a bar, all raising to the same level. For *RecPM1* and *U-AOS-FW*, population size is 168 and 262 respectively. The second figure for a function instance shows the best fitness seen as the generation progresses. To show the operator applications from each class, we have included the operator selections and best fitness progress for function instances $f04i02$, $f08i10$, $f13i04$, $f17i02$ and $f23i04$, shown in Figure 6.7.

As seen in $f05i01$, *U-AOS-FW* reaches all targets within 10 generations whereas *RecPM1* takes 18 generations. While both current-to-best/1 and best/2 operators are good operator choices to reach the targets for linear function, the faster speed is achieved with the utilisation of best/2 operator by most of the solutions though the generations. Although best/2 is one of the four operators adapted by *RecPM1*, its best utilised in the presence of other new operators included which take care of premature convergence and stagnation. For function $f07i01$, a uni modal non-separable function, *U-AOS-FW* decided to evolve solutions in the initial generations with the employment of rand-to-best/2 operator and rand/2 during the rest of the generations. It is clear from the graphs that *U-AOS-FW* reaches optimum (92.94000000000176) with much higher speed (103 generations) than *RecPM1* which manages to find solution with fitness 94.79693005303372 in 11903 generations. Thus, it becomes clear that adding new operators has improved the convergence speed by significant amount.

FIGURE 6.7: Operator application and best fitness graphs for *U-AOS-FW* (p_{min}= 0.04). Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"

*f*05*i*01



*f*07*i*01

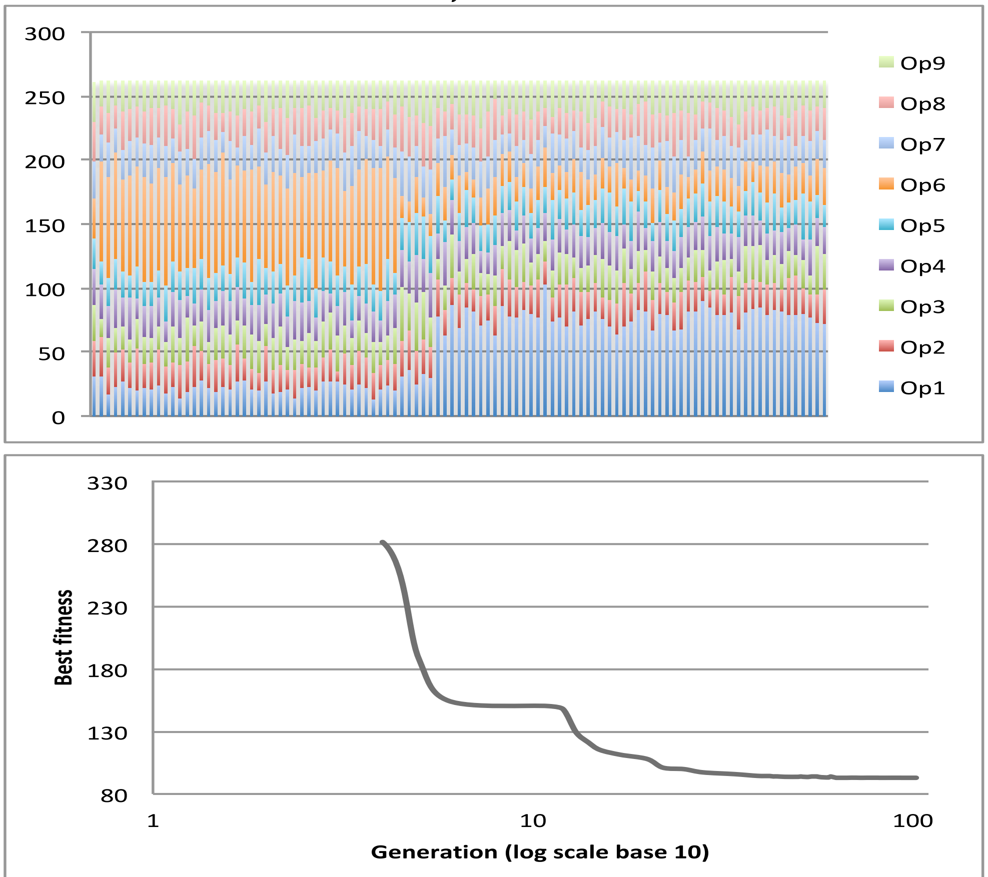
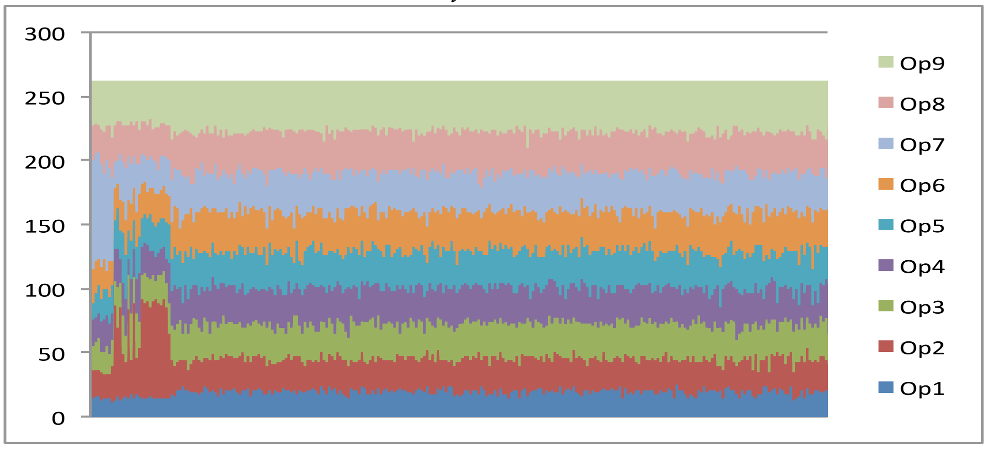
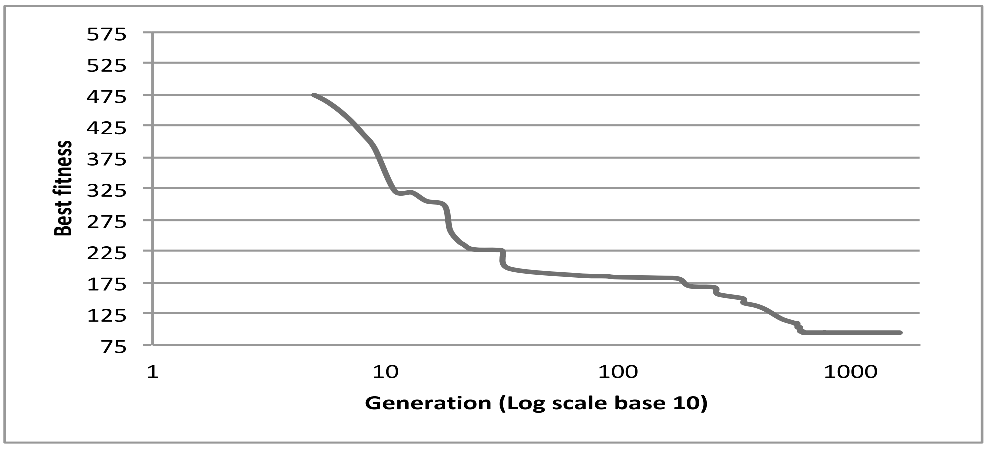


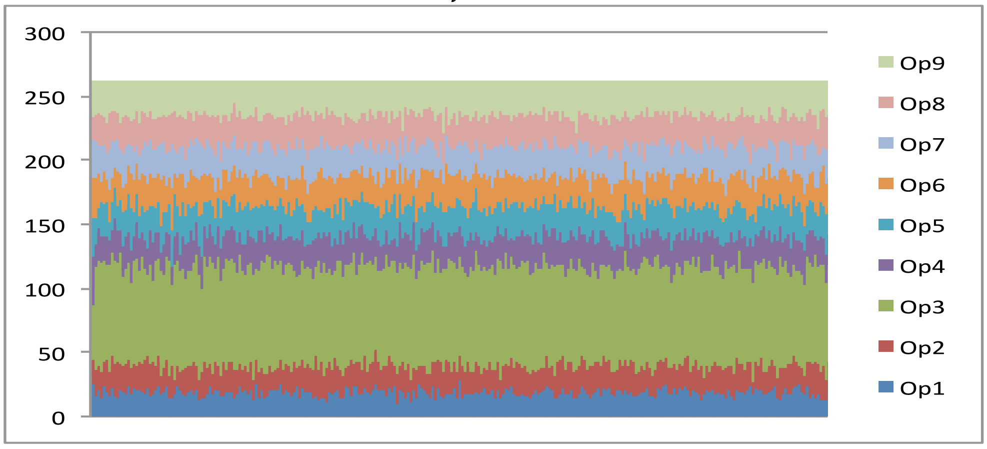
FIGURE 6.7 (cont.): Operator application and best fitness graphs for *U-AOS-FW* ($p_{\min}$= 0.04). Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"

*f*04*i*02





*f*08*i*10



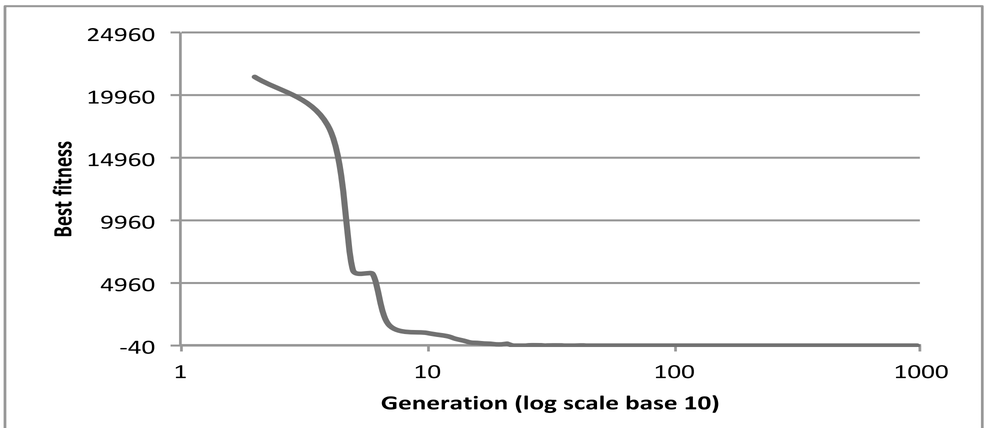
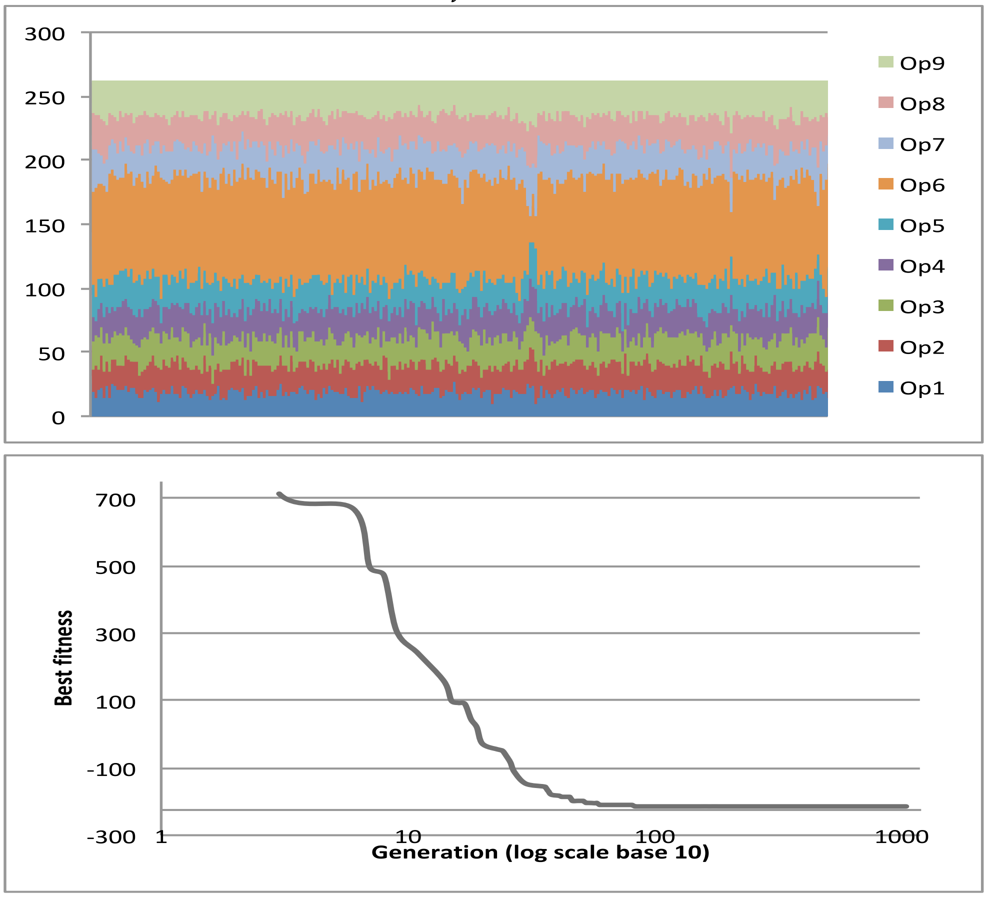


FIGURE 6.7 (cont.): Operator application and best fitness graphs for *U-AOS-FW* ($p_{\min}$= 0.04). Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"

$f13i14$



$f17i02$

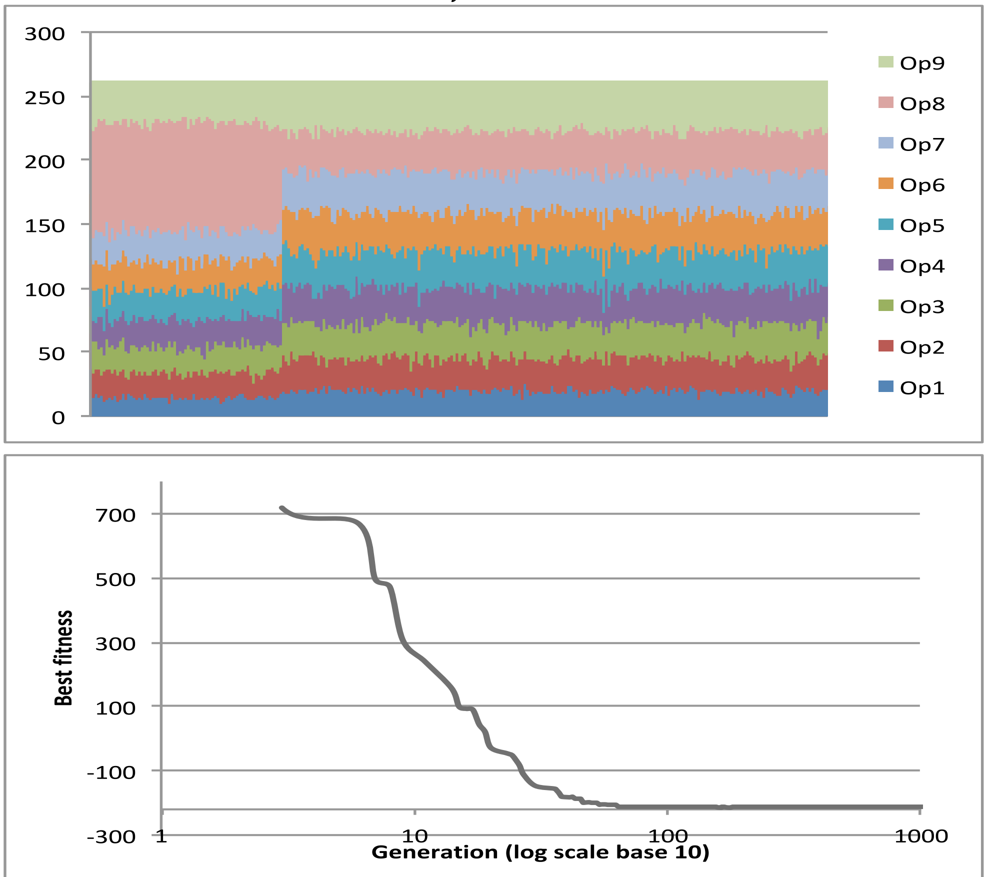
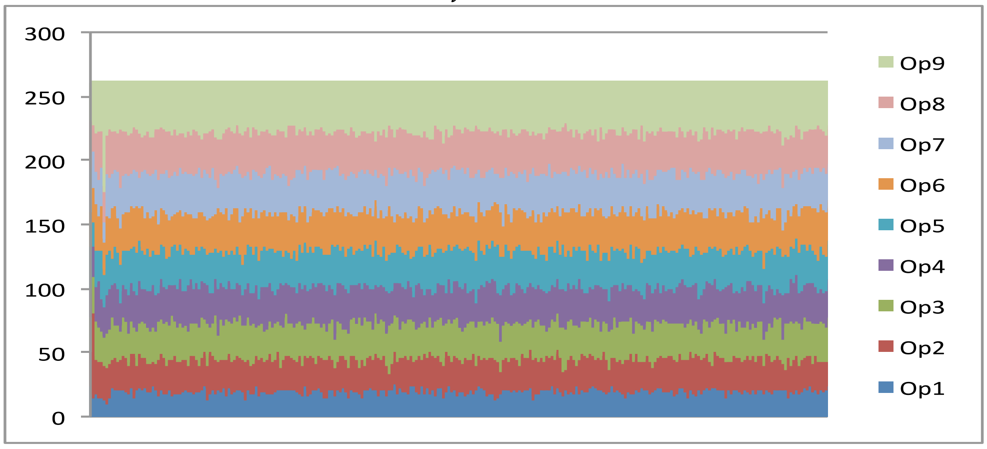
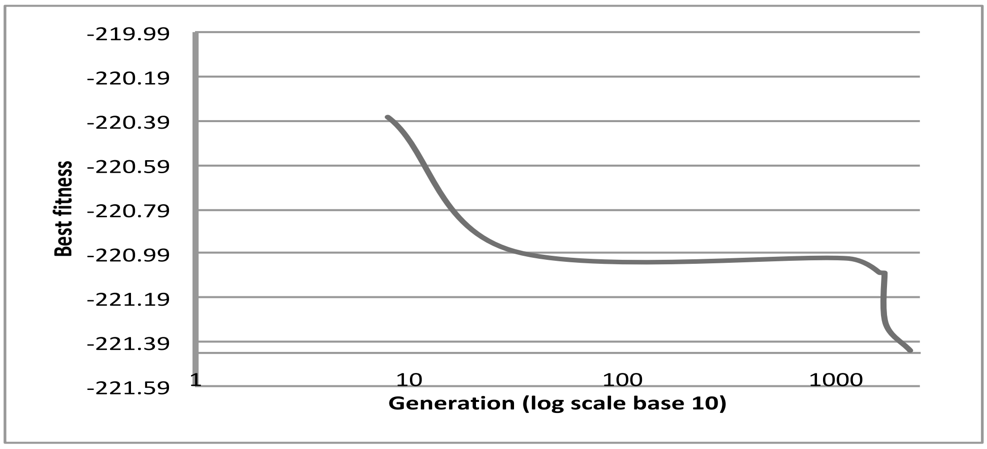


FIGURE 6.7 (cont.): Operator application and best fitness graphs for *U-AOS-FW* ($p_{\min}$= 0.04). Op1: “rand/2”, Op2: “best/1”, Op3: “current-to-best/1”, Op4: “best/2”, Op5: “rand/1”, Op6: “rand-to-best/2”, Op7: “curr-to-rand/1”, op8: “curr-to-pbest/1”, Op9: “curr-to-pbest/1(archived)”

$f23i04$





6.7 Summary

We conduct an exhaustive survey of adaptive selection of operators (AOS) in Evolutionary Algorithms (EAs). We looked at the commonality among AOS methods from the literature to combine them and we have added more components to the framework to built upon the existing categorisation of AOS methods. The formulas for the alternative choices of each component are presented. The goal is to select an AOS method from a range of AOS designs given by the presented framework such that the AOS method can learn to adaptively select the operators of differential evolution algorithm. Due to the large number of AOS choices, we employed *irace*, an offline tuner, to select an AOS method for the BBOB problem set. The set of operators consists of nine mutation strategies that have shown good performance in the literature.

irace returned a variant of *RecPM-AOS*, *U-AOS-FW*. It outperformed the four well-known *irace* tuned AOS methods, namely *Compass*, *PM-AdapSS*, *F-AUC-MAB* and *RecPM-AOS*. Among the non-AOS methods, *U-AOS-FW* outperformed *JaDE*, *CMA-ES* and *R-SHADE* in three function classes, namely low/moderate conditioning, high conditioning and multi modal classes. Overall, *U-AOS-FW* solves 5% less problems than *JaDE*. In the experiments, we showed that adaptation of carefully selected nine operators has improved the speed of convergence relative to a subset of four operators.

Chapter 7

Double Q-Network within Differential Evolution

So far AOS method is seen as an adaptation method learning with limited landscape features. This information has been broadly classified and generalised in the previous chapter. In the framework, all the information captured by different reward definitions could not be utilised at once to learn the optimal adaptation of operators. Only one reward choice is selected to convert to the probability of selection. This chapter demonstrates that the use of different landscape features and history features can be utilised at once to learn the adaptation of operators. These features are represented in a state in Reinforcement Learning (RL) [158]. An introduction to RL and literature where RL is used as a control method is shown in Chapter 3.

In RL, an agent takes an action in the environment that returns the reward and the next state. The goal is to maximize the cumulative reward at each step. Here step represents taking an action. RL estimates the value of an action given a state called *Q-value* to learn a *policy* that returns an action given a state. A variety of different techniques are used in RL to learn this policy, some of them estimating the policy indirectly and only if certain conditions are satisfied. For instance, some of the techniques are applicable only when the set of actions is finite. One such technique is Q-learning, where to obtain the best action for the current state, we first need to compute scores, known as reward, for all possible actions and then pick the one with the highest reward, making the method applicable only when the sets of state and action are finite.

In cases where states are continuous, a function is used to perform the mapping between the states and actions (as opposed to keeping an explicit map in the form of a table). This function can be approximated by a Deep Neural network (DNN) which is a neural network with more than one hidden layer. Deep Reinforcement Learning (DRL) is the setting where DNN is used to approximate the function and the weights of the network are optimized for the RL objective (to maximize the reward). Deep Q-learning, a deep reinforcement learning technique, is a specific RL setting to apply the Q-learning technique with a deep neural network as a function approximator to learn the optimal policy. In this setting, the network itself is often called Deep Q-network (DQN) [122], i.e. a deep neural network that approximates a Q-function. Deep Q-learning makes use of some tricks to ensure stability of the whole process. *Double Deep Q-Network (DDQN)* [68] is such a modification that results in more stable learning than the original DQN, using a trick that is applicable to Q-learning in general.

We present DNN used as a prediction model inside DDQN, described as an AOS method for DE. It integrates DDQN into DE as an AOS method that selects a mutation strategy for each candidate solution in each generation. Thus, action here is to select a mutation strategy out of nine operators for a parent. The efficiently in employing DDQN as a control method lies in its capability to train DDQN offline on large amounts of data and using a larger number of features to define the current state. When applied as an AOS method within DE, the proposed DE-DDQN algorithm runs many times on training benchmark problems by collecting data represented as 199 features, such as the relative fitness of the current generation, mean and standard deviation of the population fitness, dimension of the problem, number of function evaluations, stagnation, distance among solutions in the decision space, etc. After this training phase, the DE-DDQN algorithm can be applied to unseen problems. It observes the run time value of these features and predict which mutation strategy should be used at each generation.

DE-DDQN also requires the choice of a suitable reward definition to facilitate learning of a prediction model. Moreover, reward functions can be designed in different ways depending on the problem at hand. For example, Karafotias, Hoogendoorn, and Eiben [100] define and compare four per-generation reward definitions for RL-based AOS methods. As the reward definition has a strong effect on the performance of DE-DDQN, four alternative reward definitions are analysed that assign reward for each application of a mutation strategy. Some RL-based AOS methods calculate rewards per individual [131, 22], while others calculate it per generation [143]. We chose to calculate reward per individual (individual-level adaptation) as it has shown to perform good in literature.

Although in this chapter we utilise DE-DDQN to control the DE operators, it is a generic method that is applicable to control any discrete parameter in an algorithm. Thus, it can also be utilised to control for instance, crossover type and mutation type in GAs.

7.1 Methodology

We employ DQN, a DRL technique that extends Q-learning to continuous features by approximating a non-linear Q-value function of the state features using a neural network (NN). The classical DQN algorithm sometimes overestimates the Q-values of the actions, which leads to poor policies. At every step of training, the Q-network's values shift, and if we are using a constantly shifting set of values to adjust our network values, then the value estimations can easily become unstable by falling into feedback loops between the target and estimated Q-values. In order to mitigate that risk, two networks are used during training, the primary network and the target network. The target network is used to generate the target-Q values that will be used to compute the loss for every action during training. The weights of target network are fixed, and only periodically or slowly updated to the primary Q-networks values. This second network is used to generate the target-Q values. In this way training can proceed in a more stable manner. DDQN was proposed as a way to overcome this limitation and enhance the stability of the Q-values.

In our model there are large number of continuous state features. Thus, we need an implicit function approximator that maps this state to an action, as opposed to keeping an explicit map in the form of a lookup table. The mapping from state to action is termed as policy. In DRL, this function is approximated by a multi-layer perceptron neural network and the weights of the network are optimized to maximize the cumulative reward.

7.1.1 DE-DDQN

When integrated with DE as an AOS method, DDQN is adapted as follows. The environment of DDQN becomes the DE algorithm performing an optimization run for a maximum of $FE^{\max}$ function evaluations. A *state* s_t is a collection of features that measure static or run time features of the problem being solved in DE environment at step t (function evaluation). A step marks the evolution of a parent. The actions that DDQN may take are the set of mutation strategies available (Sect. 2.2.3), and a_t is the strategy selected and applied at step t. Once a mutation strategy is applied, a reward function returns the estimated benefit (or reward) r_t of applying action a_t, and the DE run reaches a new state, s_{t+1}. We refer to the tuple $\langle s_t, a_t, r_t, s_{t+1} \rangle$ as an *observation*.

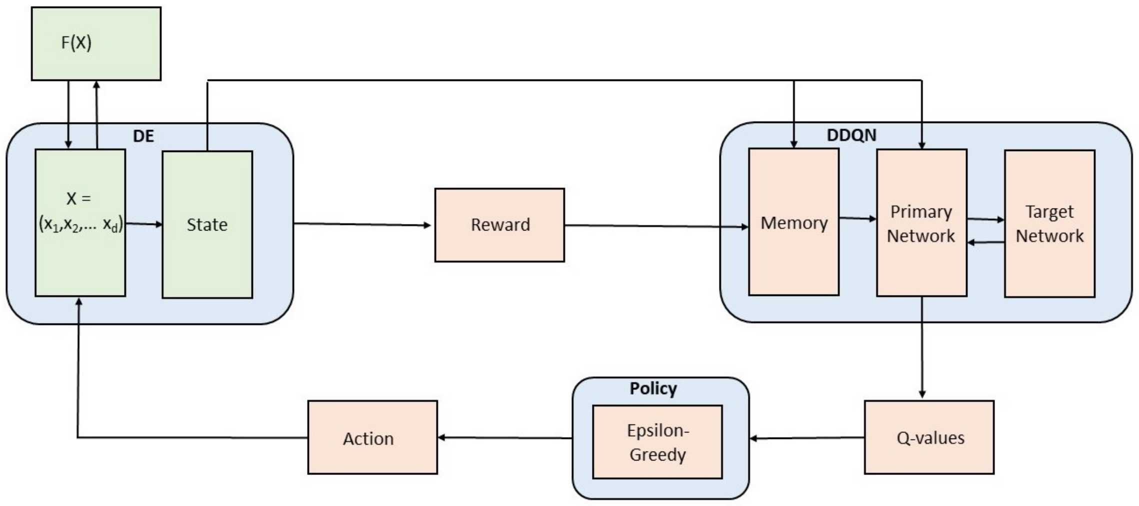
Our proposed DE-DDQN algorithm operates in two phases. In the first *training* phase, the two deep neural networks of DDQN are trained on observations by running the DE-DDQN algorithm multiple times on several benchmark functions. In the second online (or deployment) phase, the trained DDQN is used to select a mutation strategy to be applied at each generation of DE when tackling unseen test problems, different from those considered during the training phase. Next, we describe these two phases in detail.

Training phase

In the training phase, DDQN uses two deep neural networks (NNs), namely primary NN and target NN. The primary NN predicts the Q-values $Q(s_t, a; \theta)$ that are used to select an action a given state s_t at step t, while the target NN estimates the target Q-values $\hat{Q}(s_t, a; \hat{\theta})$ after the action a has been applied, where θ and $\hat{\theta}$ are the weights of the primary and target NNs, respectively, s_t is the state vector of DE, and a is a mutation strategy.

The goal of the training phase is to train the primary NN of DDQN so that it learns to approximate the target $\hat{Q}$ function. The training data is a memory of observations that is collected by running DE-DDQN several times on training benchmark functions. Training the primary NN involves finding its weights θ through gradient optimization.

FIGURE 7.1: Offline training of DE-DDQN



The training process of DE-DDQN is shown in Algorithm 6, also presented in Figure 7.1.

Training starts by running DE with random selection of mutation strategy for a fixed number of steps (*warm-up size*) that generates observations to populate a memory of capacity N, which can be different from the warm-up size (line 2). This memory stores a fixed number of N recent observations, old ones are removed as new ones are added. Once the warm-up phase is over, DE is executed M times, and each run is stopped after $FE^{\max}$ function evaluations or the known optimum of the training problem is reached (line 7). For each solution in the population, the ϵ-greedy policy is used to select mutation strategy, i.e., with ϵ probability a random mutation is selected, otherwise the mutation strategy with maximum Q-value is selected. Using the current DE state s_t, the primary NN is responsible for generating a Q-value per possible mutation strategy (line 12). The use of a ϵ-greedy policy forces the primary NN to explore mutation strategies that may be currently predicted less optimal. The selected mutation strategy is applied (line 13) and a new state s_{t+1} is achieved (line 14). A reward value r_t is computed by measuring the performance progress made at this step.

To prevent the primary NN from only learning about the immediate state of this DE run, randomly draw mini batches of observations (line 16) from memory to perform a step of gradient optimization. Training the primary NN with the randomly drawn observations helps to robustly learn to perform well in the task.

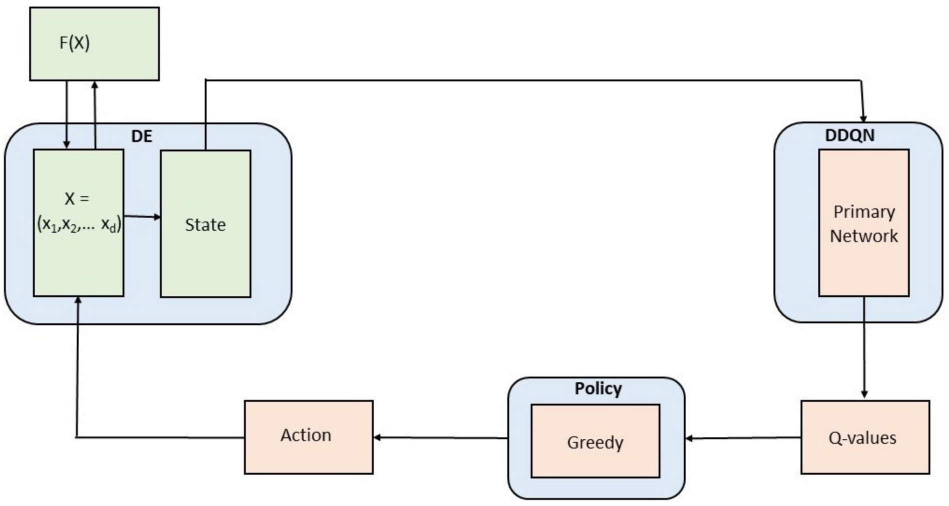
Algorithm 6 DE-DDQN training algorithm

```
1: Initialise parameter values of DE ($F$, $NP$, $CR$)
2: Run DE with random selection of mutation strategy to initialise memory to capacity $N$
3: Initialise Q-value for each action by setting random weights $\theta$ of primary NN
4: Initialise target Q-value $\hat{Q}$ for each action by setting weights $\hat{\theta} = \theta$ of target NN
5: for run $1, \ldots M$ do
6:     $t = 0$
7:     while $t < FE^{\max}$ or optimum is reached do
8:         for $i = 1, \ldots, NP$ do
9:             if rand(0, 1) < $\epsilon$ then
10:                Randomly select a mutation strategy $a_t$
11:            else
12:                Select $a_t = \arg\max_a Q(s_t, a; \theta)$
13:            Generate trial vector $u_i$ for parent $x_i$ using mutation $a_t$
14:            Evaluate trial vector and keep the best among $x_i$ and $u_i$
15:            Store observation $(s_t, a_t, r_t, s_{t+1})$ in memory
16:            Sample random mini batch of observations from memory
17:            if run terminates then
18:                $r^{\text{target}} = r_t$
19:            else
20:                $\hat{a}_{t+1} = \arg\max_a Q(s_{t+1}, a; \theta)$
21:                $r^{\text{target}} = r_t + \gamma \hat{Q}(s_{t+1}, \hat{a}_{t+1}; \hat{\theta})$
22:            Perform a gradient descent step on $(r^{\text{target}} - Q(s_j, a_j; \theta))^2$ with respect to $\theta$
23:            Every $C$ steps set $\hat{\theta} = \theta$
24:            $t = t + 1$
25: return $\theta$ (weights of primary NN)
```

The primary NN is used to predict the next mutation strategy $\hat{a}_{t+1}$ (line 20) and its reward (line 21), without actually applying the mutation. A target reward value r^{target} is used to train the primary NN, i.e., finding the weights θ that minimise the loss function $(r^{\text{target}} - Q(s_j, a_j; \theta))^2$ (line 22). If the run terminates, i.e., if the budget assigned to the problem is finished, r^{target} is the same as the reward r_t. Otherwise, r^{target} is estimated (line 21) as a linear combination of the current reward r_t and the predicted future reward $\gamma\hat{Q}(s_{t+1}, \hat{a}_{t+1})$, where $\hat{Q}$ is the (predicted) target Q-value and γ is the discount factor that makes the training focus more on immediate results compared to future rewards.

Finally, the primary and target NNs are synchronised periodically by copying the weights θ from the primary NN to the $\hat{\theta}$ of the target NN every fixed number of C training steps (line 23). That is, the target NN uses an older set of weights to compute the target Q-value, which keeps the target value r^{target} from changing too quickly. At every step of training (line 22), the Q-values generated by the primary NN shift. If we are using a constantly shifting set of values to calculate r^{target} (line 21) and adjust the NN weights (line 22), then the target value estimations can easily

FIGURE 7.2: Online process of DE-DDQN



become unstable by falling into feedback loops between r^{target} and the (target) Q-values used to calculate r^{target}. In order to mitigate that risk, the target NN is used to generate target Q-values ($\hat{Q}$) that are used to compute r^{target}, which is used in the loss function for training the primary NN. While the primary NN is trained, the weights of the target NN are fixed.

Online phase

Once the learning is finished, the weights of the primary NN are frozen. In the testing phase, the mutation strategy is selected online during an optimization run on an unseen function. The online AOS with DE is shown in the Algorithm 7, also presented in Figure 7.2.

Since the weights of the NN are not updated in this phase, we do not maintain a memory of observations or compute rewards. As a new state is observed s_t, the Q-values per mutation strategy are calculated and a new mutation strategy is chosen according to the greedy policy (line 7).

7.1.2 State features and reward

In this section we describe the new state features and reward definitions proposed for the DE-DDQN method.

Algorithm 7 DE-DDQN testing algorithm

```
1: Initialise parameter values of DE ($F$, $NP$, $CR$)
2: Initialise and evaluate fitness of each $x_i$ in the population
3: Initialise $Q(\cdot)$ for each mutation strategy with fixed weights $\theta$
4: $t = 0$
5: while $t < FE^{\max}$ do
6:     for $i = 1, \dots, NP$ do
7:         Select $a_t = \arg\max_a Q(s_t, a; \theta)$
8:         Generate trial vector $u_i$ for parent $x_i$ using operator $a_t$
9:         Evaluate trial vector $u_i$
10:        Replace $x_i$ with the best among parent and trial vector
11:        $t = t + 1$
12: return best solution found
```

State representation

The state representation needs to provide sufficient information so that the NN can decide which action is more suitable at the current step. We propose a state vector consisting of various features capturing properties of the landscape and the history of operator performance. Each feature is normalised to the range $[0, 1]$ by design in order to abstract absolute values specific to particular problems and help generalisation. Features are summarised in Tables 7.1 and 7.2.

Our state needs to encode information about how the current solutions in the population are distributed in the decision space and their differences in fitness values. The fitness of current parent $f(x_i)$ is given to the NN as a first state feature. The next feature is the mean of the fitness of the current population. The first two features in the state are normalised by the difference of worst and best seen so far solution. The third feature calculates the standard deviation of the population fitness values. Feature 4 measures the remaining budget of function evaluations. Feature 5 is the dimension of the function being solved. The training set includes benchmark functions with different dimensions in the hope that the NN are able to generalise to functions of any dimension within the training range. Feature 6, stagnation count, calculates the number of function evaluations since the last improvement of the best fitness found for this run (normalised by $FE^{\max}$).

The next set of feature values describe the relation between the current parent and the six solutions used by the various mutation strategies, i.e., the five random indexes (r_1, r_2, r_3, r_4, r_5) and the best parent in the population (x_{best}). Features 7–12

TABLE 7.1: Landscape state features

Index	Feature	Notes
1	$\frac{f(x_i) - f_{\text{bsf}}}{f_{\text{wsf}} - f_{\text{bsf}}}$	x_i denotes the i-th solution of the population and $f(x_i)$ denotes its fitness; f_{bsf} and f_{wsf} denote the best-so-far and worst-so-far fitness values found up to this step within a single run
2	$\frac{\sum_{j=1}^{NP} \frac{f(x_j)}{NP} - f_{\text{bsf}}}{f_{\text{wsf}} - f_{\text{bsf}}}$	NP is the population size
3	$\frac{\text{std}_{j=1,\ldots,NP}(f(x_j))}{\text{std}^{\text{max}}}$	$\text{std}(\cdot)$ calculates the standard deviation and std^{max} is the value when $NP/2$ solutions have fitness f_{wsf} and the other half have fitness f_{bsf}
4	$\frac{FE^{\text{max}} - t}{FE^{\text{max}}}$	FE^{max} is the maximum number of function evaluations per run, and $FE^{\text{max}} - t$ gives the remaining number of evaluations at step t
5	$\frac{dim_f}{dim^{\text{max}}}$	dim_f is the dimension of the benchmark function f being optimised, and dim^{max} is the maximum dimension among all training functions
6	$\frac{stagcount}{FE^{\text{max}}}$	*stagcount* is the *stagnation counter*, i.e., the number of function evaluations (steps) without improving f_{bsf}
7-11	$\frac{dist(x_i - x_j)}{dist^{\text{max}}}, \forall j \in \{r_1, r_2, r_3, r_4, r_5\}$	$dist(\cdot)$ is the Euclidean distance between two solutions; $dist^{\text{max}}$ is the maximum distance possible, calculated between the lower and upper bounds of the decision space; $\{r_1, r_2, r_3, r_4, r_5\}$ are random indexes
12	$\frac{dist(x_i - x_{\text{best}})}{dist^{\text{max}}}$	x_{best} is the best parent in the current population
13-17	$\frac{f(x_i) - f(x_j)}{f_{\text{wsf}} - f_{\text{bsf}}}, \forall j \in \{r_1, r_2, r_3, r_4, r_5\}$	
18	$\frac{f(x_i) - f(x_{\text{best}})}{f_{\text{wsf}} - f_{\text{bsf}}}$	
19	$\frac{dist(x_i - x_{\text{bsf}})}{dist^{\text{max}}}$	x_{bsf} denotes the solution with fitness f_{bsf}

TABLE 7.2: History state features

Index	Feature	Notes
20-55	$\sum_{g=1}^{\max_{\text{gen}}} \frac{N_m^{\text{succ}}(g,op)}{N^{\text{tot}}(g,op)}$	For each *op* and $m \in \{1,2,3,4\}$ and normalised over all operators; $\max_{\text{gen}}$ is the number of recent generations recorded; $N_m^{\text{succ}}(g,op)$ and $N^{\text{tot}}(g,op)$ are successful and total applications of *op* according to OM_m at generation g
56-91	$\frac{\sum_{g=1}^{\max_{\text{gen}}} \sum_{k=1}^{N_m^{\text{succ}}(g,op)} OM_m(g,k,op)}{\sum_{g=1}^{\max_{\text{gen}}} N^{\text{tot}}(g,op)}$	
92-127	$\frac{OM_m^{\text{best}}(g,op) - OM_m^{\text{best}}(g-1,op)}{OM_m^{\text{best}}(g-1,op) \cdot \lvert N^{\text{tot}}(g,op) - N^{\text{tot}}(g-1,op)\rvert}$	For each *op* and $m \in \{1,2,3,4\}$ and normalised over all operators; $OM_m^{\text{best}}(g,op)$ is the maximum value of $OM_m(g,k,op)$ in current generation g
128-163	$\sum_{g=1}^{\max_{\text{gen}}} OM_m^{\text{best}}(g,op)$	For each *op* and $m \in \{1,2,3,4\}$ and normalised over all operators
164-199	$\sum_{w=1}^{W} OM_m(w,op)$	For each *op* and $m \in \{1,2,3,4\}$ and normalised over all operators; $OM_m(w,op)$ is the w-th value in the window generated by *op*

measure the Euclidean distance in decision space between the current parent x_i and the six solutions. These six euclidean distances help the NN learn to select the strategy that best combines these solutions. Features 13–18 use the same six solutions to calculate the fitness difference w.r.t. $f(x_i)$. Feature 19 measures the normalised Euclidean distance in decision space between x_i and the best solution seen so far. We use distances instead of positions to make the state representation independent of the dimensionality of the solution space.

Describing the current population is not sufficient to select the best strategy. Reinforcement learning requires the state to be Markov, i.e., to include all necessary information for selecting an action. To this end, we enhance the state with features about the run time history. Using historical information has shown to be useful in our previous work [150], presented in Chapter 5. In addition to the remaining budget and the stagnation counter described above, we also store four metric values $OM_m(g, k, op)$ after the application of op at generation g:

1. $OM_1(g, k, op) = f(x_i) - f(u_i)$, that is, the k-th fitness improvement of offspring u_i over parent x_i;
2. $OM_2(g, k, op)$, the k-th fitness improvement of offspring over x_{best}, the best parent in the current population;
3. $OM_3(g, k, op)$, the k-th fitness improvement of offspring over x_{bsf}, the best so far solution; and
4. $OM_4(g, k, op)$, the k-th fitness improvement of offspring over the median fitness of the parent population.

For each OM_m, the total number of fitness improvements (*successes*) is given by $N_m^{\text{succ}}(g, op)$, that is, the index k is always $1 \leq k \leq N_m^{\text{succ}}(g, op)$. The counter $N^{\text{tot}}(g, op)$ gives the total number of applications of op at generation g. We store this historical information for the last $\max_{\text{gen}}$ number of generations.

With the information above, we compute the sum of success rates over the last $\max_{\text{gen}}$ generations, where each success rate is the number of successful applications of operator op, i.e., mutation strategy, in generation g that improve metric OM_m divided by the total number of applications of op in the same generation. For each metric OM_m, the values for an operator are normalised by the sum of all values of all operators. A different success rate is calculated for each combination of OM_m ($m \in \{1, 2, 3, 4\}$) and *op* (nine mutation strategies) resulting in features 20–55.

We also compute the sum of fitness improvements for each OM_m divided by the total number of applications of *op* over the last $\max_{\text{gen}}$ generations (features 56–91). Features 92–127 are defined in terms of best fitness improvement of a mutation strategy *op* according to metric OM_m over a given generation g, that is, $OM_m^{\text{best}}(g, op) = \max_k^{N_m^{\text{succ}}(g,op)} OM_m(g, k, op)$. In this case, we calculate the relative difference in best improvement of the last generation with respect to the previous one, divided by the difference in number of applications between the last two generations (g and $g - 1$). Any zero value in the denominator is ignored. The sum of best improvement seen for combination of operator and metric is given as features 128–163. Features 164-199 are calculated by maintaining a fixed size window W where each element is a tuple of the four metric values $OM_m, m \in \{1, 2, 3, 4\}$ and $f(u_i)$ resulting from the application of a mutation strategy to x_i that generates u_i. Initially the window is filled with OM_m values as new improved offspring are produced. Once it is full, new elements replace existing ones generated by that mutation strategy according to the First-In First-Out (FIFO) rule. If there is no element produced by that operator in the window, the element with the worst (highest) $f(u_i)$ is replaced. Each feature is the sum of OM_m values within the window for each m and each operator. The difference between features extracted from recent generations (128-163) and from the fixed-size window (164-199) is that the window captures the best solutions for each operator, and the number of solutions present per operator vary. In a sense, solutions compete to be part of the window. Whereas when computing features from the last $\max_{\text{gen}}$ generations, all successful improvements per generation are captured and there is no competition among elements. As the most recent history is the most useful, we use small values for last $\max_{\text{gen}} = 10$ generations and window size $W = 50$.

Reward definitions

While we only know the true reward of a sequence of actions after a full run of DE is completed, i.e., the best fitness found, such sparse rewards provide a very weak signal and can slow down training. Instead, we calculate rewards after every action has been taken, i.e., a new offspring u_i is produced from parent x_i. In this chapter, we explore four reward definitions, each one using different information related to fitness improvement except the last one that calculates the cumulative area of a problem instance:

$$\text{R1} = \max\{f(x_i) - f(u_i), 0\} \qquad \text{R2} = \begin{cases} 10 & \text{if } f(u_i) < f_{\text{bsf}} \\ 1 & \text{else if } f(u_i) < f(x_i) \\ 0 & \text{otherwise} \end{cases}$$

$$\text{R3} = \max\{\tfrac{f(x_i)-f(u_i)}{f(u_i)-f_{\text{optimum}}}, 0\} \qquad \text{R4} = \text{The area under the cumulative curve}$$

R1 is the fitness difference of offspring from parent when an improvement is seen. This definition has been used commonly in literature for parameter control [131, 22, 143]. R2 assigns a higher reward to an improvement over the best so far solution than to an improvement over the parent. R3 is a variant of R1 relative to the difference between the offspring fitness and the optimal fitness, i.e., maximise the fitness difference between parent and offspring and minimise fitness difference between offspring and optimal solution. This definition can only be used when the optimum values of the functions used for training are known in advance. Finally, R4 takes into account the target of a function achieved by DE-DDQN. We consider 51 targets from 10^{-8} to 10^{2} given a budget. This reward definition takes into account the speed of an algorithm. The more the targets are achieved with higher speed, the greater the reward is. The least reward is attained when no target is reached. This can be an indication of stagnation or premature convergence.

7.2 Experimental Design

In the implementation of DE-DDQN, the primary and target NNs are multi-layer perceptrons. We integrate the four reward definitions R1, R2, R3 and R4 into DE-DDQN and the resulting methods are denoted as DE-DDQN1, DE-DDQN2, DE-DDQN3 and DE-DDQN4, respectively. For each of these methods, we trained four NNs using batch sizes 64 or 128 and 3 or 4 hidden layers, and we picked the best combination

TABLE 7.3: Hyper-parameter values of DE-DDQN

Training and online parameters	**Parameter value**
Scaling factor (F)	0.5
Crossover rate (CR)	1.0
Population size (NP)	100
$FE^{\max}$ per function	10^4 function evaluations
Max. generations ($\max_{\text{gen}}$)	10
Window size (W)	50
Type of neural network	Multi layer perceptron
Hidden layers	4
Hidden nodes	100 per hidden layer
Activation function	Rectified linear (Relu) [125]
Batch size	64
Training only parameters	**Parameter value**
Training policy	ϵ-greedy ($\epsilon = 0.1$)
Discount factor (γ)	0.99
Target network synchronised (C)	every $1e3$ steps
Observation memory capacity	10^5
Warm-up size	10^4
NN training algorithm	Adam (learning rate: 10^{-4})
Online phase parameters	**Parameter value**
Online policy	Greedy

of batch size and the number of hidden layers according to the total accumulated reward during the training phase. In all cases, the most successful configuration was batch size 64 with 4 hidden layers. The rest of the parameters are not tuned but set to default values. In the training phase, we applied ϵ-greedy policy with $\epsilon = 10\%$, that means 10% of the actions are selected randomly and the rest according to the highest Q-value. In the warm-up phase during training, we set the capacity of the memory of observations larger than the warm-up size so that 90% of the memory is filled up with observations from random actions and the rest with actions selected by the NN. The gradient descent algorithm used to update the weights of the NN during training is Adam [105]. Table 7.3 shows all hyper-parameter values.

7.2.1 Training and Testing

As an experimental benchmark, we use functions from the BBOB test bed. In order to force the NN to learn a general policy, we train on different classes of functions from BBOB. For BBOB details refer to Chapter 4.2.1. In particular, the proposed DE-DDQN method is first trained on 48 out of 360 function instances from five function classes for dimension 20. Two random instances from each 24 functions are included in the training set. Then, we run the trained DE-DDQN on a different set of 15 functions, also for dimension 20. Three functions are selected randomly from each of the five classes, thus a total of 15 test functions.

During training, we cycle through the 48 training problems multiple times and keep track of the mean reward achieved in each cycle. We overwrite the weights of the NN if the mean reward is better than what we have observed in previous cycles. We found this measure of progress was better than comparing rewards after individual runs, because different problems vary in difficulty making rewards incomparable. After each cycle, the 48 problems are shuffled before being used again. This is done for DE-DDQN not able to learn in a fixed pattern and make it to learn on problems seen in random order. The mean reward stopped improving after approximately 105 cycles (5040 problems, 5040×20^4 FEs), 103 cycles (4944 problems, 4944×20^4 FEs), 102 cycles (4896 problems, 4896×20^4 FEs) and 131 cycles (6288 problems, 6288×20^4 FEs) for DE-DDQN1, DE-DDQN2, DE-DDQN3, DE-DDQN4 respectively which indicated the convergence of the learning process.

Although the computational cost of the training phase is significant compared to a single run of DE, this cost is incurred offline, i.e., one time on known benchmark functions before solving any unseen function, and it can be significantly reduced by means of parallelisation and GPUs. On the other hand, we conjecture that training on even more data from different classes of functions should allow the application of DE-DDQN to a larger range of unknown functions.

After training, the NN weights were saved and used for the testing (online) phase.[1] For testing, each DE-DDQN variant was run on each test problem shown

[1]The weights obtained after training are available on Github [**https://github.com/mudita11/DE-DDQN-bbob/**] together with the source code, and can be used for testing on similar functions. The code may be adapted to train or test using other benchmark suites such as CEC2014 with functions of up to dimension 40.

TABLE 7.4: Test set for DE-DDQN. $fxiy$ denotes a function instance iy that is obtained by a transformation of original function fx.

Function class	**Function instance**
Separable functions	$f01i15$, $f02i01$, $f03i15$, $f04i02$, $f05i01$
Function with low or moderate conditioning	$f06i01$, $f07i01$, $f08i10$, $f09i13$
Functions with high conditioning and uni modal	$f10i07$, $f11i08$, $f12i14$, $f13i14$, $f14i06$
Multi modal functions with adequate global structure	$f15i11$, $f16i10$, $f17i02$, $f18i06$, $f19i10$
Multi modal functions with weak global structure	$f20i08$, $f21i10$, $f22i10$, $f23i04$, $f24i10$

in Table 7.4 and each run was stopped when either absolute error difference from the optimum is smaller than 10^{-8} or $10^5 \times 20$ function evaluations are exhausted. ECDF graphs for each function is shown in Figure 7.3.

We compared the four presented DE-DDQN variants with three baselines: *U-AOS-FW*, *JaDE* and *R-SHADE*. We do not compare the performance of proposed algorithms with *F-AUC-MAB*, *PM-AdapSS*, *Compass* and *RecPM-AOS* as *U-AOS-FW* performed better than these in the Chapter 6. The parameters of *U-AOS-FW* is tuned in the same manner as described in Chapter 6 with the help of an offline configurator *irace*.

7.3 Experiments and Results

We present ECDF graphs of four DE-DDQN variants and three algorithms in comparison. The aRT tables are shown in Appendix C.

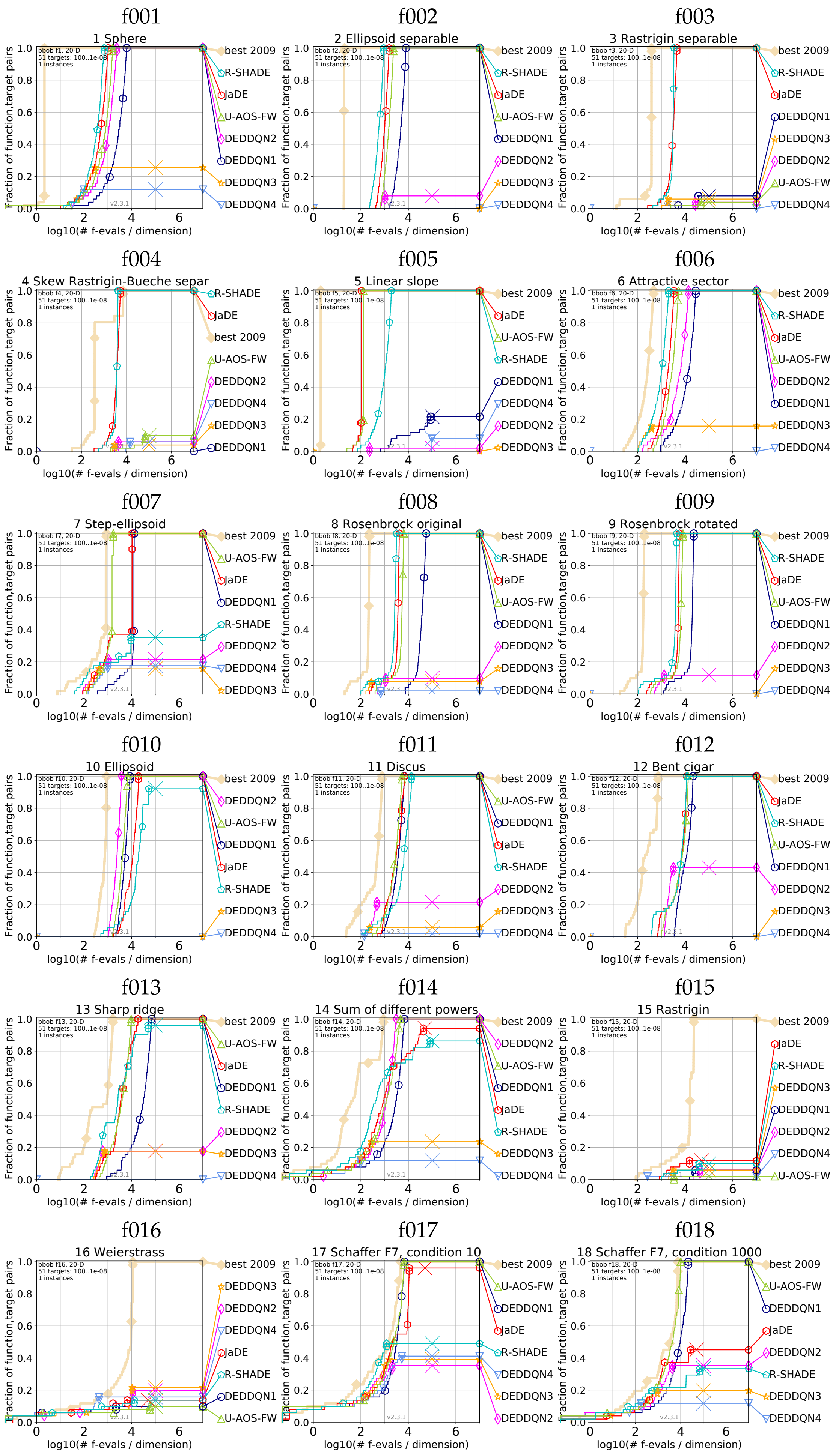


FIGURE 7.3: ECDFs on test set

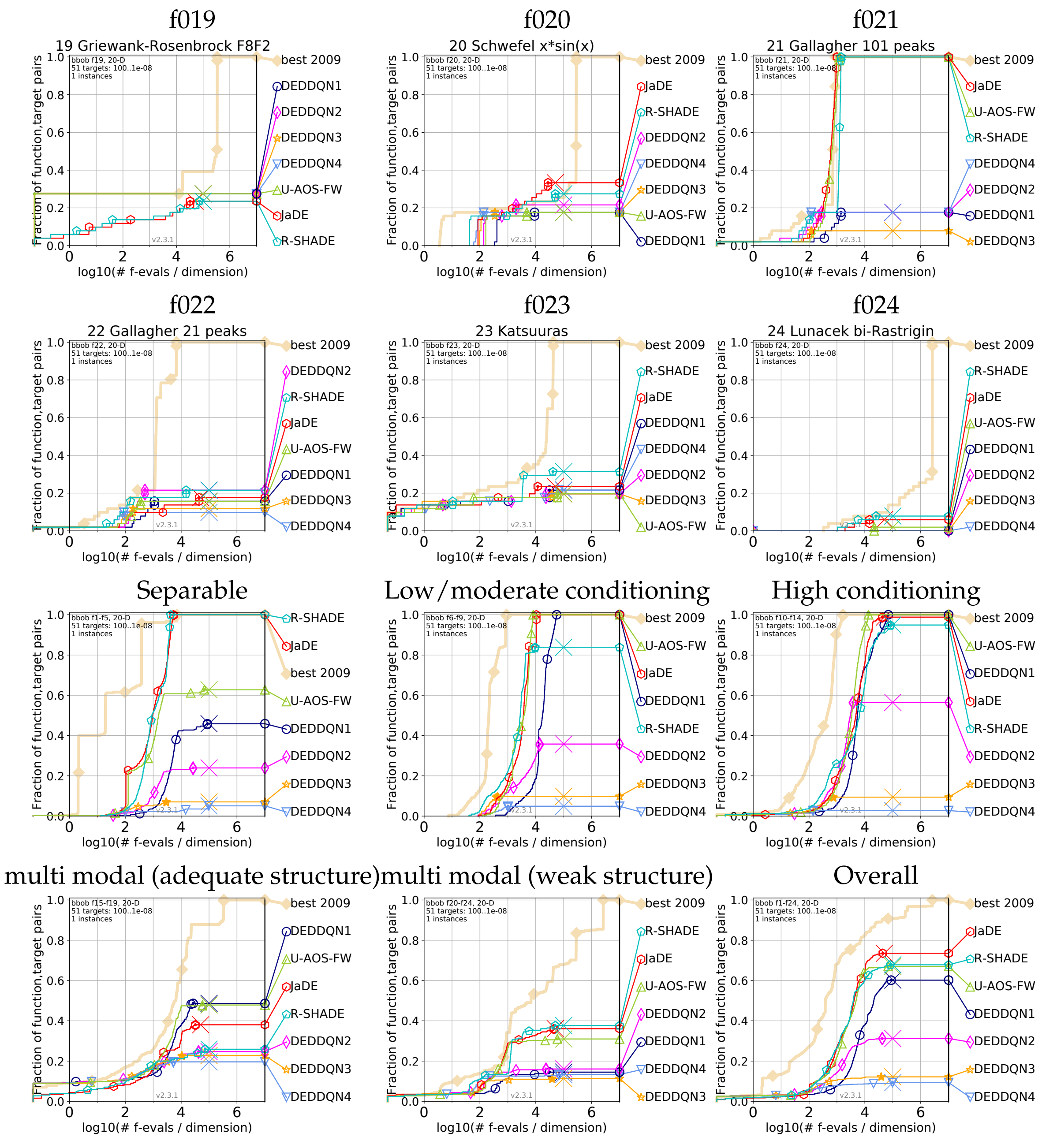


FIGURE 7.3 (cont.): ECDFs on test set

7.3.1 Comparison among four proposed models

DE-DDQN1 has performed better than other three variant of presented DE-DDQN algorithm for thirteen out of 24 test instances. In particular, it reached all targets for test instances from function $f6$ to $f14$. In addition to these instances, it also reached all targets for function instances $f1$, $f2$, $f17$ and $f18$. Among other instances, it could not achieve all targets but shows better or at par performance with either of DE-DDQN2, DE-DDQN3 and DE-DDQN4 for $f3$, $f5$, $f15$, $f19$, $f21$, $f23$ and $f24$. DE-DDQN2 outperformed others in functions $f4$, $f20$ and f$f22$. DE-DDQN3 showed best result for function $f16$. Thus, DE-DDQN4 could not outperform any other proposed variants of DE-DDQN. Overall, DE-DDQN1 solved 30% more targets than DE-DDQN2 which outperforms DE-DDQN2 and DE-DDQN4.

Reward definitions play an important role in deciding the action for the next state. They should be informative enough to facilitate learning in the dynamic environment. R1 and R2 have shown potential in deciding the operator selection for various solutions in the search space where the functions have same dimension ranges $[-5,5]^{20}$. R1 considers the improvement of offspring over parent and can be seen as rewarding the actions proportional to the improvements made by them. This has been an effective definition but due to varying step-sizes, it is a slow approach. R2 is a simple definitions that assigns fix reward values and does not get affected by the function range. It assigns ten times more reward when offspring improves over the best so far solution than when it improves over its parent. Thus, DE-DDQN2 has shown improvement for few functions by learning to generate offspring that not only tend to improve over the parent but also improve the best fitness seen so far. The R2 definition encourages the generation of better offspring than the best so far candidate and it is invariant to differences in function ranges. Although DE-DDQN1 and DE-DDQN2 has shown potential in learning adaptation, they suffer from the known issue of generalisation. We employ RL algorithms with the aim to generalise on functions with broad properties. However, generalisation has been highlighted as an active issue in DRL and is under investigation [128]. The issue of overfitting leads to the lack of generalisation. As pointed out in [24] "agents become overly specialised to the environments encountered during training" leading to the problem of overfitting. This has been realised in the context of gaming environment. In the evolutionary environments, due to lack of utilisation of deep RL methods for adaptation, there is no analysis in literature to deal with the issue of generalisation. Thus, we identify this issue and take it as a future work that make a step towards mitigating overfitting problem by including more functions in the training set [44]. [87] identifies that RL agents regularly overfit to a particular training distribution. This can be seen in our environment where a DE-DDQN variant performs good on certain functions only. As can be seen in ECDF graphs 7.3, in some cases ($f5$, $f8$, $f12$ etc.) DE-DDQN1 performs better than other DE-DDQN variants and for functions such as $f20$, $f6$ DE-DDQN2 outperforms others. Like R1, R3 also involve raw function values. Although R1 is less informative than R3, which considers improvement over parent and optimum value, latter has shown poor performance. Although R3 scales fitness improvement with distance from the optimum which partially mitigates the effect of different ranges among functions with consistent ranges, it is ranked third among four variants of DE-DDQN. The improvement can be small or large when functions with different ranges are considered. As a result, R3 becomes less efficient about choosing operators that will solve the problem within the given number of function evaluations. It is surprising that R4 definition has been most ineffective to learn the online selection of operators. The main idea behind designing this definition is to improve the convergence speed of the algorithm. It however could not learn simple fitness landscapes and would need future analysis.

7.3.2 Comparison of proposed models with other algorithms

As DE-DDQN1 outperformed other DE-DDQN variants, we compare the performance of DE-DDQN1 with other algorithms in comparison. DE-DDQN1 achieves all targets for instances involved in low/moderate conditioning, high conditioning and multi modal (adequate structure) classes of functions. However, it lagged in speed in comparison to *JaDE* and *U-AOS-FW*. It outperformed all algorithms in comparison for multi modal (adequate structure) class.

7.3.3 Comparison of operator selection by DE-DDQN1 and *U-AOS-FW*

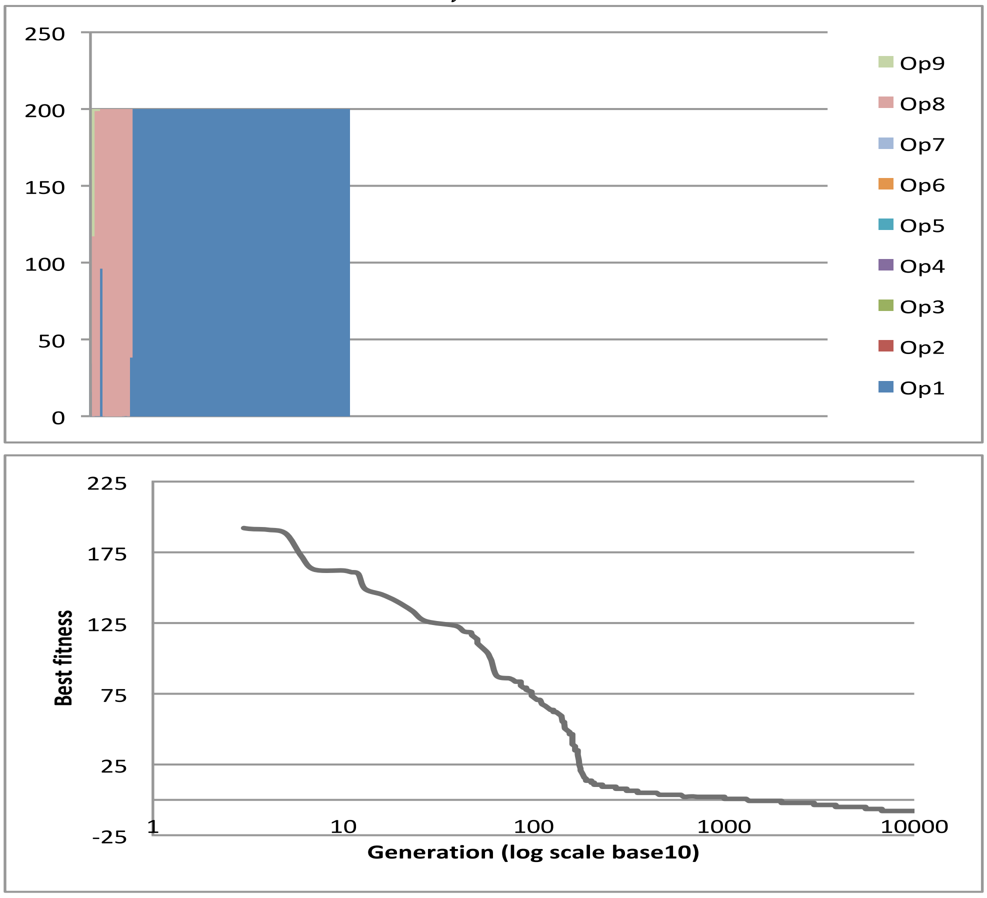
To identify the differences in the framework setting returned by *irace* (Chapter 6) and the best variant of DE-DDQN (DE-DDQN1), we observe the pattern in the operator selection made by DE-DDQN1 and *U-AOS-FW* shown in Figures 6.7 and D.3 respectively. These figures also show the evolution of best fitness in different generations for a randomly selected function instance from each class from the test bed. The operator applications for these functions selected by DE-DDQN2, DE-DDQN3 and DE-DDQN4 are shown in appendix C.

Although both DE-DDQN1 and *U-AOS-FW* have utilised different operators in different generations of the function instances, a notable selection pattern can be seen in these figures. DE-DDQN1 adapts operators by applying one operator to evolve all parents in a generation and switching to another operator only after a number of generations. A total of two operators out of nine is a common choice for DE-DDQN1 in the whole run of DE. On the contrary, *U-AOS-FW* utilises more than one operator in a generation, that is it employs different operators to evolve different parents in a generation. Also, the number of applications of each operator is non-uniform in different generations.

The graph for function instance $f23i04$ makes it clear that DE-DDQN1 performed better than *U-AOS-FW* because DE-DDQN1 learned to utilise different operators in different generations whereas *U-AOS-FW* has uniformly applied each operator throughout the generations. Precisely, during the initial two-third generations, DE-DDQN1 employs current-to-best/1 with other operators for perturbation and towards the end applying operator rand/2. The function for which both algorithms have shown same performance ($f17i02$), has one aspect in common. They both utilised current-to-pbest/1 operator for the initial one-third generations. However, during the rest of the generations, *U-AOS-FW* utilised all operators uniformly and DE-DDQN1 applied only current-to-pbest/1(archived) operator. Thus, it implies that current-to-pbest/1(archived) gave same performance as uniform selection of operators during the last generations. For functions $f13i14$, $f8i10$ and $f7i01$, DE-DDQN1 and *U-AOS-FW* reach all targets but the former with lower speed. It can be concluded that *U-AOS-FW* learned to utilise all operators to enhance the speed as DE-DDQN1

FIGURE 7.4: Operator application and best fitness graphs for DE-DDQN1. Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"

$f05i01$



$f07i01$

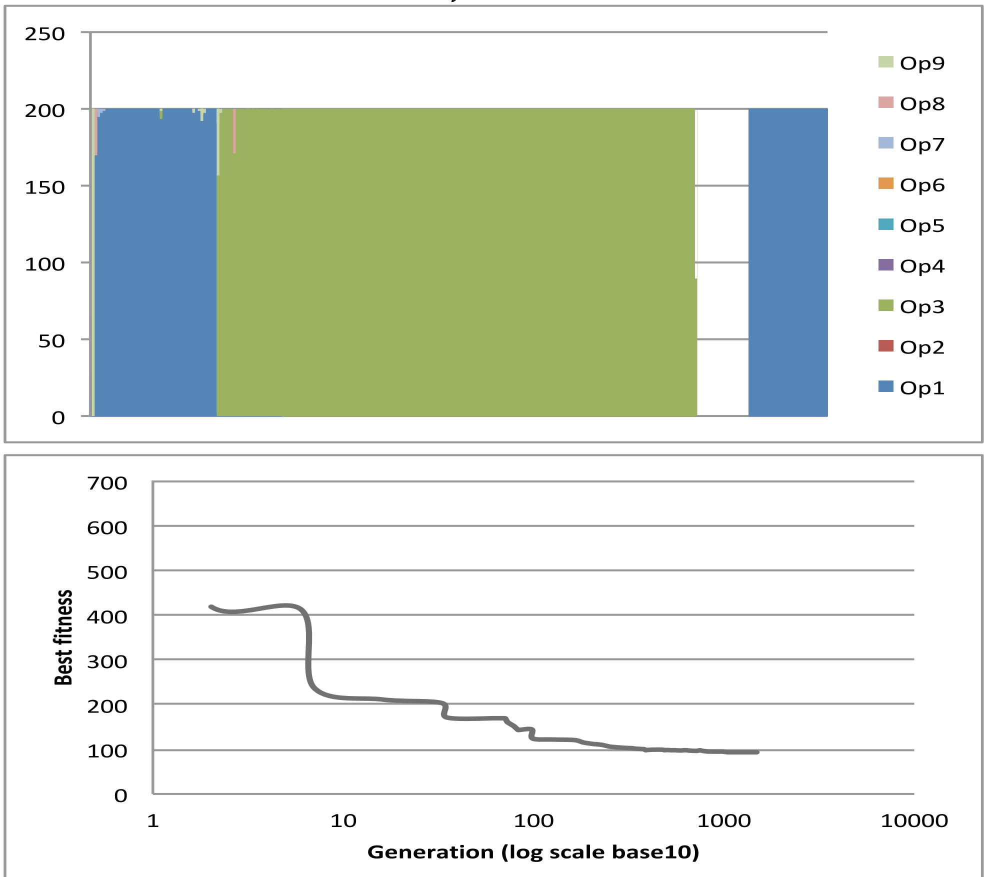
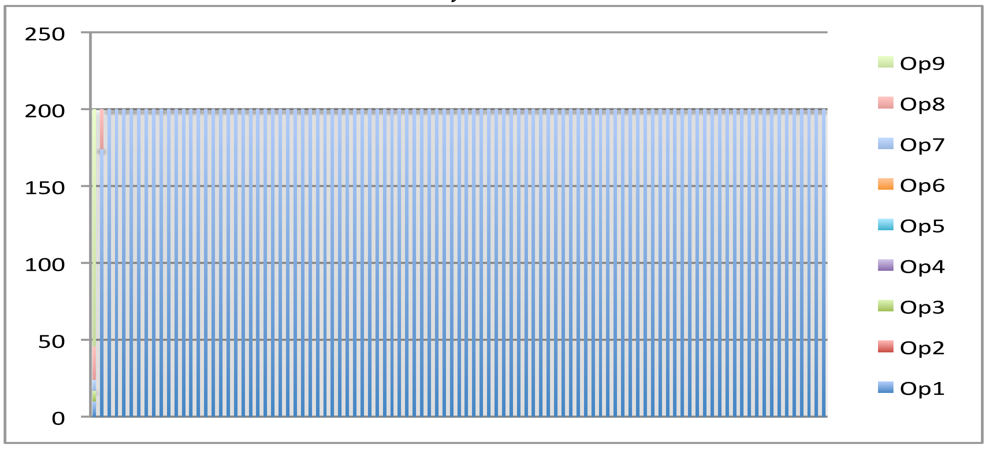
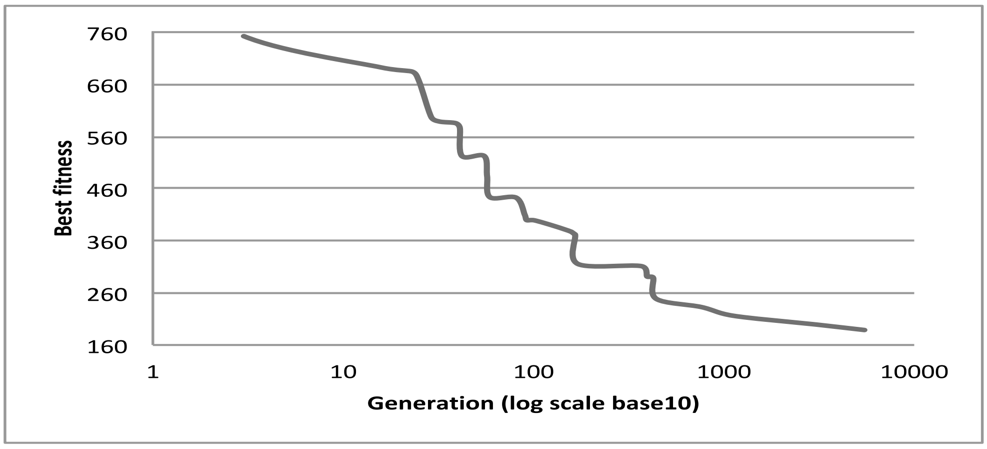


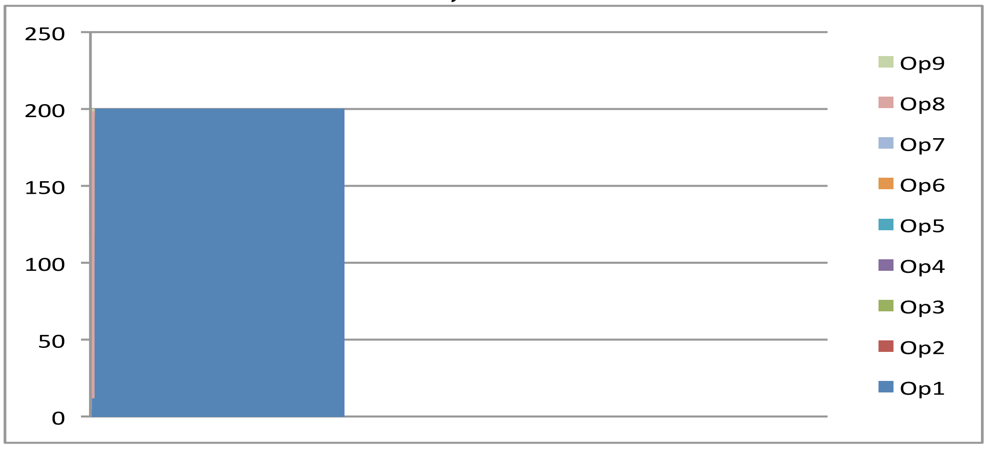
FIGURE 7.4 (cont.): Operator application and best fitness graphs for DE-DDQN1. Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"

$f04i02$





$f08i10$



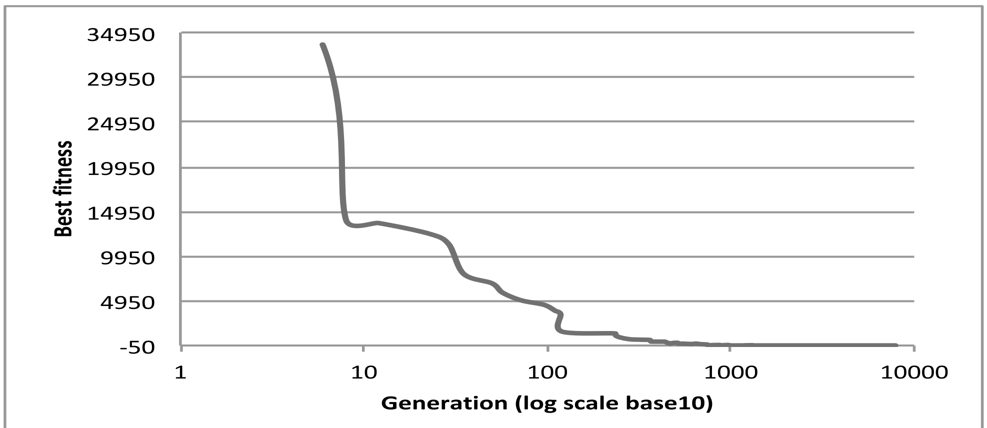
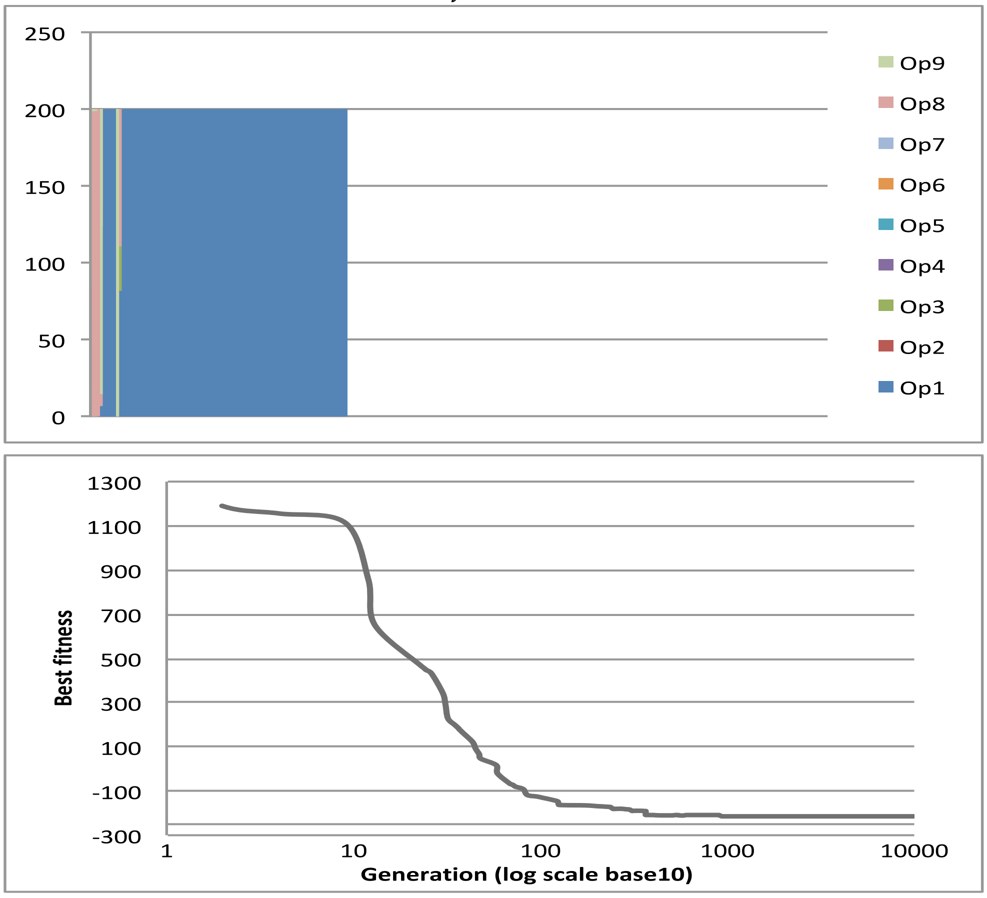


FIGURE 7.4 (cont.): Operator application and best fitness graphs for DE-DDQN1. Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"

$f13i14$



$f17i02$

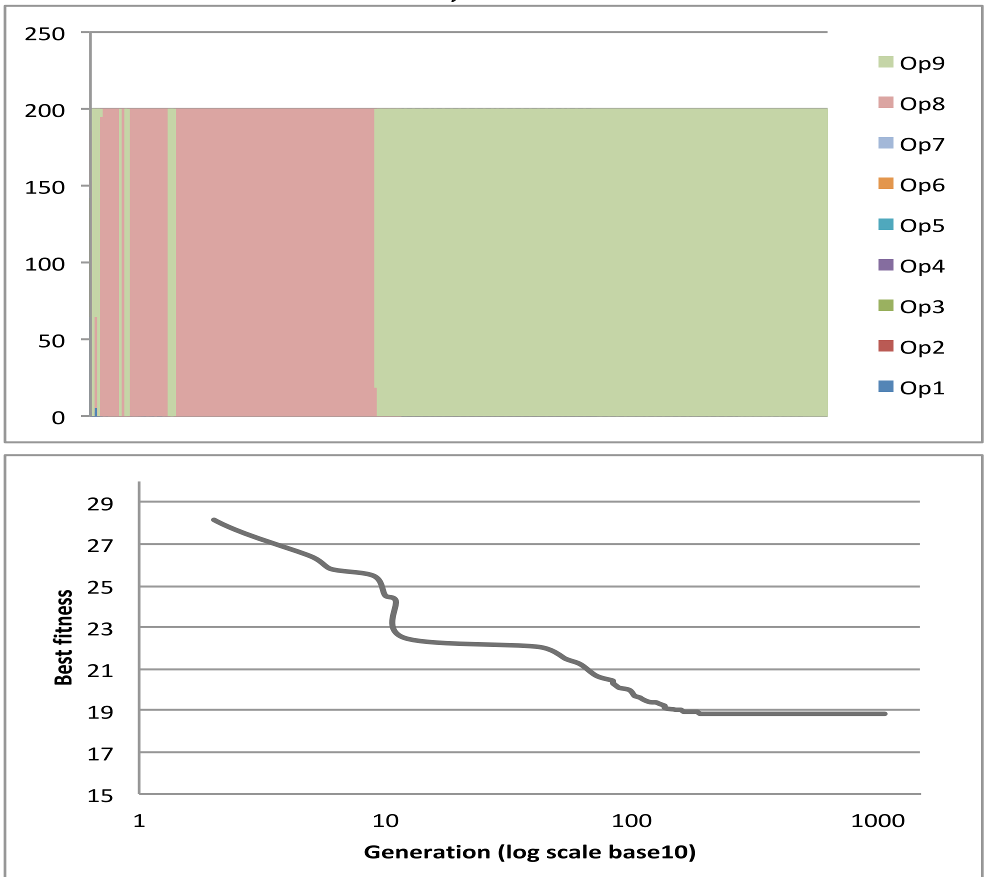
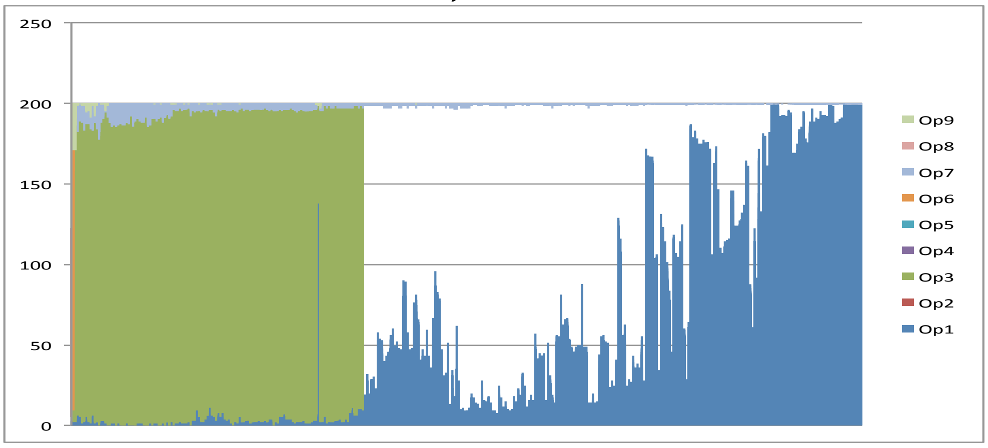
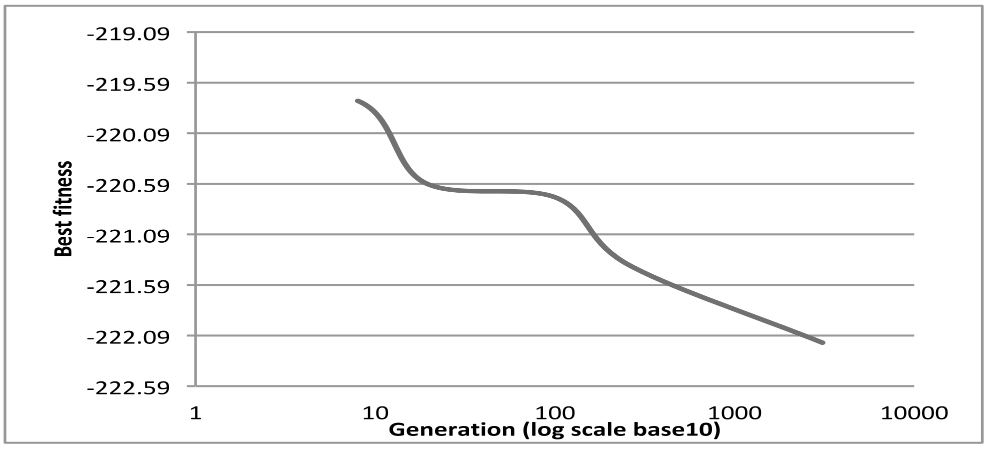


FIGURE 7.4 (cont.): Operator application and best fitness graphs for DE-DDQN1. Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"

*f*23*i*04





hardly used more than two operators for achieving different targets in these functions. DE-DDQN1 performed poorly for function instance $f04i02$ and $f05i01$ compared to *U-AOS-FW* due to less utilisation of different operators.

7.4 Summary

In this chapter, we proposed an AOS method based on Double Deep Q-Learning (DDQN), a Deep Reinforcement Learning method, to control the mutation strategies of Differential Evolution (DE). The application of DDQN to DE requires two phases. First, a neural network is trained offline by collecting data about the DE state and the benefit (reward) of applying each mutation strategy during multiple runs of DE tackling benchmark functions. We define the DE state as the combination of 199 different features and we analyze four alternative reward functions. Second, when DDQN is applied as a parameter controller within DE to a different test set of benchmark functions, DDQN uses the trained neural network to predict which mutation strategy should be applied to each parent at each generation according to the DE state. The benchmark functions for training and testing are taken from the BBOB benchmark set with dimension 20. We compare the results of the four proposed DE-DDQN variants to one DE based AOS algorithm *U-AOS-FW*, two DE based non-AOS algorithms.

The results show that DE-DDQN1 outperforms the other DE-DDQN variants for all function instances in the test set. It reaches all targets for functions instances considered in low/moderate conditioning and high conditioning classes. It showed best performance for multi modal (adequate structure) class. It, however could not perform overall best due to moderate performance in separable and multi modal (weak structure) class. This is a result of overfitting that prevents the algorithm to generalise well on large set of functions.

Part IV

Conclusion and Future Work

Chapter 8

Conclusion

Evolutionary Algorithms (EAs) are stochastic algorithms which are inspired by natural processes. These algorithms have solved various real world problems efficiently. The performance depends largely on the selection of the parameters. Thus, tuning parameters of an algorithm becomes an important task as it gives an insight on the performance of the algorithm. The optimum parameter values for a problem can be different for different problems. Various parameter tuners and controllers are discussed in Chapter 2. However, adaptive mechanisms deciding the selection of optimal parameter values at different stages of the algorithm are required for efficient performance of an EA. Adaptive Operator Selection (AOS) is an approach that controls the discrete parameters during the run of an EA. Heuristic and reinforcement learning based approaches are discussed in Chapter 3.

This thesis has presented various methods for the adaptive selection of mutation strategies in Differential Evolution (DE). We proposed AOS methods that select an operator for a parent on-the-fly. These AOS methods have been combined with an offline tuner to show their applicability together. Online and offline tuning are performed at different stages of the whole procedure. We have utilised an existing offline tuner *irace*. *irace* has been efficiently utilised to tune the parameters involved in presented AOS, while AOS adapts the operators of DE. For testing the algorithms, we chose single objective toy problems from BBOB noiseless benchmark set. Section 8.1 summarises various contributions presented in the thesis toward the research questions defined in Chapter 1. Lastly, section 8.2 gives possible directions of future work.

8.1 Summary of contributions

Chapter 5 has answered the first research question to be able to efficiently utilise the markov reward processes as an adaptive method. The proposed algorithm, *RecPM-AOS*, has been utilised for the selection of four operators in DE. The goal is to maximise the future cumulative reward attained in each generation. As *RecPM-AOS* has its own hyper-parameters, we make three sets of comparisons for the fair assessment. First, the hyper-parameters of *RecPM-AOS* and parameters of DE are trained on the BBOB training set using *irace*. Second, the hyper-parameters of *RecPM-AOS* are tuned using *irace* and the parameters of DE are set as default. Third, all the parameters of *RecPM-AOS* and DE are set as default. These three set of experiments are also performed with other popular AOS methods from the literature. The *RecPM-AOS* variant with all tuned parameters outperformed all the variants of AOS methods including its other variants, showing the efficiency of employing offline tuner at a level that does not directly impact the solutions of numerical functions.

Although many AOS methods have been proposed and their performance has been compared with other algorithms, the field of evolutionary computation lacks a generic framework that is widely accepted by the community. Chapter 6 achieves the aims of the second research question where we designed a unified framework of AOS methods that can be utilised to control the discrete parameters of any algorithm. This framework builds upon the existing classification of AOS methods by further simplifying its taxonomy. The presented framework consists of various AOS components and their generalised choices. This framework has been utilised to replicate existing methods and to design many new ones. We employed *irace* to select an AOS method from a large set of AOS designs formed by combining different component choices present in the framework. It also tunes the hyper-parameters of selected AOS method along with the parameters of DE. In this chapter, we included five more strategies in the set of four strategies from Chapter 5 to analyse the role of new added strategies in balancing exploration and exploitation. The experiments are conducted on BBOB problem set to learn the online selection of nine operators. *irace* was given a population of starting configurations and it returned a variant of *RecPM-AOS*, named *U-AOS-FW*. *U-AOS-FW* outperforms *RecPM-AOS* trained with nine and four operators. The addition of five more operators helped perturbation of solutions throughout the generations and improved speed over four operators. However, the aRT tables for AOS methods within DE algorithm showed that no one algorithm has best converging speed for all functions.

Chapter 7 utilises deep reinforcement learning with the aim to learn from a large number of state features. These features result from landscape and history information of nine DE operator applications. This work is conducted towards achieving the target of the third research question. Four reward definitions have been proposed, out of which two showed promising results. The same approach is applied on CEC2005 problem set in Appendix B. The testing of presented algorithm, DE-DDQN, is conducted on two benchmark sets. One, where all functions have same dimensional ranges (BBOB) and the other, that has different ranges for different functions (CEC2005) has given a clue that one reward definition might perform better than the other based in the inconsistency of the function ranges considered in the problem set. This approach can be utilised for online selection of discrete parameter belonging to any evolutionary algorithm with the aim to generalise on functions with broad set of properties. However, overfitting on training set seems to be a common issue and needs to be dealt with in the future.

Comparing the performance of all proposed methods, it can be concluded that beside improving the performance of an algorithm, online control of parameters impacts the speed of convergence. The methods which learned the optimal adaptive behaviour achieve function targets with faster speed than those who fail to learn the optimal adaption or end up utilising just one to two operators for the whole run.

8.2 Future work

In Chapter 6 we presented a unified framework of AOS methods. The framework consists of a large number of AOS designs. Thus, we decided to give *irace* a population of starting configurations. *irace* returns different configurations each time the starting configurations are changed. As the resultant configuration depends on the population of starting configurations, there is need for analysis for the careful selection of these configurations.

We identify a number of possible future works in the scope of deep reinforcement learning based DE-DDQN which are expected to improve the performance of the method. DRL algorithms have a drawback that they overfit on the training problem set, thus does not generalise well on the testing set. One possible way to prevent overfitting and improve generalisation is by including a larger training set. As mentioned in Chapter 4, tuning in DE-DDQN could not have been possible using *irace* as it is a time-consuming process. There is a need to explore the tuners such as bayesian optimsation for extensive tuning of state features and hyper-parameter values. This tuning process should identify the most relevant feedback information to enclose in a state within a feasible time frame.

There are other parameters in the DE algorithm that impacts the performance of DE. *JaDE* is an efficient algorithm that controls the scaling factor (F) and crossover rate (CR) in DE with a modified mutation strategy. One effective approach could be to utilise the control methods from *JaDE* or from machine learning (specifically reinforcement learning) within framework and DE-DDQN for controlling various DE continuous parameters.

The proposed methods are flexible enough to be extended to control discrete parameters of other evolutionary algorithms such as *SHADE* and *CMA-ES*. Thus, to improve the performance of existing evolutionary algorithms, the presented control methods can be employed to adapt discrete parameters.

Part V

Appendices

Appendix A

Mutated Artificial Bee Colony algorithm

A.1 Introduction

Hybridisation of evolutionary algorithms (EAs) has been used by researchers to overcome the drawbacks of population-based algorithms. One of the issue with evolutionary algorithms is that they suffer from premature convergence. The Artificial Bee Colony (ABC) [88] algorithm is one such algorithm that is good at exploitation but lacks exploration capability. An introduction to ABC can be found in Chapter 2.2.4. It is proposed by Dervis Karaboga in 2005 that is based on the imitation of the foraging behaviour of bees. It uses intelligent behaviour of a swarm of bees and has successfully solved a range of mathematical problems [147]. To improve the exploitation and exploration capability of ABC, we present a novel variant of ABC named mutated Artificial Bee Colony algorithm. To achieve this we found new parameters of ABC and tune them manually to solve various numerical problems. ABC search can be compared to mutation in GA and thus we employ the properties of mutation into ABC. In addition to this, we replace the criteria to replace scout bee in the scout bee phase. It helps to prevent abandoning good solutions from the population with possibility to introduce the new better ones. Thus, these steps resulted into mutated ABC that works towards improving the search capability of ABC. The proposed algorithm is compared with various standard EAs and popular ABC variants on four classes of numerical toy problems.

Further sections are organised as follows: Section 2 discusses the proposed variant of ABC algorithm. Section 3 gives experimental results and finally, Section 5 provides conclusions.

A.2 Methodology

The standard ABC algorithm consists of three phases after initialisation phase. These are employed bee phase, onlooker bee phase and scout bee phase. They are responsible for exploitation and exploitation of the search space. Detailed working of ABC can be found in Chapter 2.2.4. We present proposed algorithm known as mutated artificial bee colony (mutated ABC) algorithm in this section. Its working is shown in the Algorithm 8. The proposed algorithm starts by initialising the parameters. The generally known parameters of ABC are population size (NP) and ($limit$) that determines the number of scout bees. In addition to these, we identify new parameters involved in mutated ABC which are as follows, ratio of the number of employed bees to the number of onlooker bees ($EB : OB$), selection method of an employed bee (EB) in the employed bee phase ($Select_{EB}$), selection method of an onlooker bee (OB) in the onlooker bee phase ($Select_{OB}$) and random number generator (RNG). The new potential position of a bee falls within a range defined by RNG. In ABC, the $EB : OB$ is taken as 1:1 that is the number of employed bees are same as the number of onlooker bees. $Select_{EB}$ and $Select_{OB}$ is the selection method used to evolve selected employed and onlooker bees respectively. All EB and OB are selected to evolve to produce a new candidate position (or solution) in ABC. ABC considers RNG as rand$[-1, 1]$ that can take a random number within -1 and 1. These parameters play an important role in the working of the algorithm. Thus, we made an attempt to manually set the parameter values according to best suit the four class of problems considered.

In the Initialization phase, all bees in the colony are assigned a random position within their dimensional limits. Further steps are repeated until a termination condition is satisfied. An employed bee, x_i, explores two new position in the direction of a selected k-th employed bee, x_k. It is selected using binary tournament selection mechanism. To enhance the search capability, we divide the original equation 2.2

Algorithm 8 Mutated Artificial Bee Colony algorithm

```
1: Initialise parameter values of ABC: $NP$, $limit$, $EB : OB$, $\mu_p$, $Select_{EB}$, $Select_{OB}$,
   $RNG$
2: Randomly initialise the position of bees
3: Evaluate the fitness of each individual in the population
4: $g = 0$ (generation number)
5: while stopping condition is not satisfied do
6:     for each $\vec{x}_i, i = 1, \ldots, NP/2$ do
7:         Generate two positions using equations A.1
8:         Greedy selection of a position
9:     for each $\vec{x}_i, i = NP/2, \ldots, NP$ do
10:        Roulette-wheel selection of an employed bee using equation 2.3
11:        Generate two positions using equations A.1
12:        Greedy selection of a position
13:    Randomly assign a position to the worst bee
14:    $g = g + 1$
```

involved in the employed bee phase into two equations shown below A.1. The difference between the original and proposed equations is the difference in the RNG parameter, one takes the RNG as rand$[-1, 0]$ and other as rand$[0, -1]$.

$$v_i = x_i + \text{rand}[-1, 0](x_i - x_k), k \in [1, NP], i \neq k \tag{A.1}$$

$$v_i' = x_i + \text{rand}[0, 1](x_i - x_k), k \in [1, NP], i \neq k \tag{A.2}$$

According to these two equations, v_i and v_i' is a position of x_i bee which is towards and away from the k-th bee. That is, v_i and v_i' fall in the opposite directions of x_i.

Another difference is the introduction of mutation rate μ_P. These equations can be compared to mutation in Genetic Algorithms (GAs). Like in mutation in GAs we flip one or more bits according the mutation rate, the real values in a position vector are changed according the introduced parameter μ_P. In ABC, only one randomly selected dimension value of a bee position is changed, whereas in mutated ABC we assign each dimension a probability of getting selected for mutation. We name this probability as mutation rate, inspired from GAs. The assignment of a selection probability to each dimension of a position, enhances the search capability in the mutated ABC.

In the employed bee phase, each EB chooses the best position among x_i and positions resulting from the above two equations, v_i and v_i' based on their fitness. In the onlooker bee phase, an OB exploits the neighbourhood of selected employed bee.

The selection of an employed bee is made probabilistically (that is roulette wheel selection) from the employed bee present in the current population, as shown in equation 2.3. The neighbourhood of the EB is exploited using equation A.1 where x_i is the selected EB and x_k is the randomly selected EB. At the end of this phase each OB attains a position that is best among its current position, v_i and v_i'. Thus in both phases of mutated ABC, each bee has a choice to either attain a position towards or away from the k-th bee and the best position is chosen according to their fitness.

An employed bee whose position has not been changed for a predefined number of generations becomes a scout bee. The number of scout bees in a generation is determined by parameter $limit$. If the value of $limit$ parameter is set as a large value, there is a possibility to lose an employed bee with good fitness value. If $limit$ value is small, it can result in slow convergence speed. In ABC, the value of $limit$ is set as either 0 or 1. We keep the value of $limit$ same as in ABC, however we propose that an EB becomes scout bee only if its fitness is greater than the population average fitness (in case of minimisation). As a result at the end of a generation its position is perturbed as in employed bee phase and not initialised randomly. This is different from standard ABC where a new random position is assigned without considering the 'goodness' of the current position that can lead to a new position farther away from optima.

A.3 Experimental results

The experiments are conducted on a set of benchmark functions shown in Table A.1. Each function belongs to one of the following class of problem: Uni modal Separable (US), Multi modal Separable (MS), Uni modal Non-separable (UN), and Multi modal Non-separable (MN). Each problem was carried out for 30 runs and each run consists of $500,000$ function evaluations. The population size is kept fixed to 50 (NP) in each generation leading to $500000/50 = 10,000$ generations in a run. The fitness of a solution in the search space with values less than 10^{-15} is taken as 0. The class, dimension (Dim), Interval and optimum value for all the functions is shown in Table A.1. These specifications are matched with paper [95].

TABLE A.1: Test problems

Function name	**Class**	**Dim**	**Interval**	**Global minima (F_{min})**
Sphere	US	30	[-100, 100]	0
Rosenbrock	UN	30	[-30, 30]	0
Rastrigin	MS	30	[-5.12, 5.12]	0
Griewank	MN	30	[-600, 600]	0
Schaffer	MN	2	[-100, 100]	0
Dixon-Price	UN	30	[-10, 10]	0
Ackley	MN	30	[-32, 32]	0
Schwefel	MS	30	[-500, 500]	-12569.5
SixHump-CameBack	MN	2	[-5, 5]	-1.03163
Branin	MS	2	[-5, 10] $\times$ [0, 15]	0.398

We set the value of $EB : OB$ as in the original ABC algorithm that is 1:1. The number of scout bee (*limit*) is atmost 1 in all runs for all experiments. Mutation probability (μ_p) is set to 0.01 for all functions except for Dixon-Price (Uni modal Non-separable) function whose mutation probability is 1.0 which means that each dimension of an employed bee undergoes change as it needs more exploration of the search space. This value is attained by manual tuning. $Select_{EB}$, $Select_{OB}$ are kept same as in ABC that is all bees are search for new positions in the employed and onlooker bee phase.

We compare the performance of mutated ABC with six other algorithms: Genetic Algorithm (GA)[140], Particle Swarm Optimisation (PSO) [37], Differential Evolution (DE) [132], ABC [88], quick ABC (qABC) [94] and adaptive and hybrid ABC (aABC) [95]. The working of GA, DE and ABC is explained in Chapter 2. PSO is a well-known optimisation algorithm that moves towards the optimum solution by changing the velocity of the candidate solutions towards global best and local best solution. qABC [94] modifies onlooker bee phase where each onlooker bee chooses a neighbour according to the euclidean distance similarity measure rather than proportionate selection. It has significant improvement over standard ABC algorithm. [95] proposes aABC as an adaptive and hybrid ABC algorithm for the adaptive network fuzzy inference system. It employs crossover rate and adaptivity coefficient in aABC that increases the convergence speed and improves the performance of ABC.

Table A.2 shows the results obtained with the proposed algorithm. The mean, the standard deviation, the best solution and the worst solution from 30 runs are shown along with last column showing best results obtained with aABC. Bold entries show the best results obtained by an algorithm for each function. Thus, mutated ABC obtains best solutions for US, MS and MN class functions compared to aABC algorithm. Near-optimal solutions are obtained for UN class of functions. Mutated ABC finds optimal solution in all 30 runs for sphere, Rastrigin, Griewank, Schaffer and Ackley functions. This can be said because for these functions mean lies on the optima with 0 standard deviation.

We compare the mean values generated by each algorithm for each function shown in Table A.3. The values for six algorithms are taken directly from paper [95] for comparison. The results obtained by best algorithm for a function is marked in bold. All algorithms except GA manage to find optimal solution on Sphere function, a uni modal separable function, in all runs. aABC has outperformed all algorithms for the two uni modal non-separable functions, Rosenbrock and Dixon-Price, with ABC also finding optimum solution for Dixon-Price function. Comparing results for multi modal separable functions, it can be noticed from Table A.3 that mutated ABC, aABC, qABC and ABC manages to find optimum solution on Rastrigin and Schwefel. These three algorithms could find a near-optimal solution for Branin function unlike GA that finds the optimal solution. Considering last class which is multi modal Non-separable includes Griewank, Schaffer, Ackley and SixHump-CameBack, it is noticed that ABC and all its variants have shown best mean results for Griewank function. For Schaffer function, PSO, DE, aABC and mutated ABC have outperformed other algorithms. For Ackley function, DE, ABC along with all variants of ABC have shown best results. Lastly, for SixHumpCameBack, GA has outperformed all other algorithms.

TABLE A.2: Results on test problems

Function name	**Mean**	**Standard Deviation**	**Worst**	**Best-mutated ABC**	**Best aABC**
Sphere	0	0	0	**0**	**0**
Rosenbrock	0.085543	0.127395	0.681654	0.001961	**2.1913E-005**
Rastrigin	0	0	0	**0**	**0**
Griewank	0	0	0	**0**	**0**
Schaffer	0	0	0	**0**	**0**
Dixon-Price	0.000116504	0.000367662	0.001937097	1.44909E-08	**2.2822E-015**
Ackley	0	0	2.84E-14	**0**	2.2204E-014
Schwefel	-12569.487	5.55E-12	-12569.487	**-12569.487**	**-12569.487**
SixHump-CameBack	-1.0316284	0	-1.0316284	**-1.0316284**	**-1.0316284**
Branin	0.409121	0.020377	0.482377	**0.3978949**	0.398874

TABLE A.3: Mean result on benchmark functions using GA, PSO, DE, ABC, qABC, aABC and mutated ABC

Function name	**GA**	**PSO**	**DE**	**ABC**	**qABC**	**aABC**	**Mutated ABC**
Sphere	1.11E+03	**0**	**0**	**0**	**0**	**0**	**0**
Rosenbrock	1.96E+05	15.088617	18.203938	0.1766957	0.1329198	**0.0246333**	0.085543
Rastrigin	52.92259	43.977137	11.716728	**0**	**0**	**0**	**0**
Griewank	10.63346	0.0173912	0.0014792	**0**	**0**	**0**	**0**
Schaffer	0.004239	**0**	**0**	1.04E-10	8.66E-06	**0**	**0**
Dixon-Price	1.22E+03	0.6666667	0.6666667	**0**	1.15E-12	**0**	1.91E-02
Ackley	14.67178	0.1646224	**0**	**0**	**0**	**0**	**0**
Schwefel	-11593.40	-6909.1359	-10.266	**-12569.49**	**-12569.49**	**-12569.49**	**-12569.49**
SixHump-CameBack	**-1.03163**	-1.0316285	-1.031628	-1.0316284	-1.0316284	-1.0316284	-1.0316284
Branin	**0.397887**	0.3978874	0.3978874	0.3978874	0.3978874	0.3978874	0.409121

A.4 Conclusion and summary

In this chapter, a novel variant of Artificial Bee Colony algorithm has been proposed which explores new parameters of ABC and tunes few of them to enhance exploration ability of ABC. The mutation operator borrowed from Genetic Algorithm has proved useful by interpolating and extrapolating the position of bees in finding new better solutions in their neighbourhood. Replacing 'limit' parameter with the average fitness of bees has been successful in perturbing position of employed bee and finding global minima. Mutated ABC is tested on four classes of problems and shown best results for Uni modal Separable, Multi modal Separable, Multi modal Non-separable function and near optimal solution for Uni modal Separable Non-separable function. The results are compared with six other population-based algorithms, namely Genetic Algorithm, Particle Swarm Optimsation, Differential Evolution, standard Artificial Bee Colony algorithm and its two variants- quick Artificial Bee Colony algorithm and adaptive Artificial Bee Colony algorithm. Overall results show that mutated ABC is at par with aABC and better than other above-mentioned algorithms. The novel algorithm is best suited to 3 of the 4 classes of functions under consideration. Functions belonging to UN class have shown near optimal solution.

Appendix B

DE-DDQN tested on CEC2005 problem set

We present results on CEC2005 problem set using algorithm DE-DDQN presented in Chapter 7. We train it on four operators "best/1", "best/2", "curr-to-best/1" and "rand/2", shown in section 2.2.3. Thus, there are four actions that DE-DDQN learns to adapt while running Differential Evolution (DE) algorithm. There are 19 landscape features as shown in Table 7.1 and due to four actions, there are 80 history based features shown in Table B.1. Thus, state consists of 99 features in total.

We consider three models in this chapter. DE-DDQN combined with R1, R2 and R3 reward definitions, discussed in section 7.1.2. We name these models as DE-DDQN1, DE-DDQN2 and DE-DDQN3. Next we discuss the results of experiments on CEC2005 problem set.

B.1 Experimental design

The hyper-parameter setting is same as shown in Table 7.3. We compare the DE-DDQN1, DE-DDQN2 and DE-DDQN3 with ten baselines: random selection of mutation strategies (Random), four different fixed-strategy DEs (DE1-DE4), *PM-AdapSS* (AdapSS) [49], *F-AUC-MAB* (FAUC) [59], *RecPM-AOS* (RecPM) [150] and the two winners of CEC2005 competition, which are both variants of *CMA-ES*: LR-CMAES (LR) [11] and IPOP-CMAES (IPOP) [10]. For details on DE-DDQN variants see Chapter 7. Among all these alternatives, *PM-AdapSS*, *F-AUC-MAB*, *RecPM-AOS* are

TABLE B.1: History state features

Index	**Feature**	**Notes**
20-35	$\sum_{g=1}^{\max_{\text{gen}}} \frac{N_m^{\text{succ}}(g,op)}{N^{\text{tot}}(g,op)}$	For each op and $m \in \{1,2,3,4\}$ and normalised over all operators; $\max_{\text{gen}}$ is the number of recent generations recorded; $N_m^{\text{succ}}(g,op)$ and $N^{\text{tot}}(g,op)$ are successful and total applications of op according to OM_m at generation g
36-51	$\frac{\sum_{g=1}^{\max_{\text{gen}}} \sum_{k=1}^{N_m^{\text{succ}}(g,op)} OM_m(g,k,op)}{\sum_{g=1}^{\max_{\text{gen}}} N^{\text{tot}}(g,op)}$	
52-67	$\frac{OM_m^{\text{best}}(g,op) - OM_m^{\text{best}}(g-1,op)}{OM_m^{\text{best}}(g-1,op) \cdot \lvert N^{\text{tot}}(g,op) - N^{\text{tot}}(g-1,op)\rvert}$	For each op and $m \in \{1,2,3,4\}$ and normalised over all operators; $OM_m^{\text{best}}(g,op)$ is the maximum value of $OM_m(g,k,op)$
68-83	$\sum_{g=1}^{\max_{\text{gen}}} OM_m^{\text{best}}(g,op)$	For each op and $m \in \{1,2,3,4\}$ and normalised over all operators
84-99	$\sum_{w=1}^{W} OM_m(w,op)$	For each op and $m \in \{1,2,3,4\}$ and normalised over all operators; $OM_m(w,op)$ is the w-th value in the window generated by op

AOS methods that were proposed to adaptively select mutation strategies. The parameters of these AOS methods were previously tuned with the help of an offline configurator *irace* [115] and the tuned hyper-parameter values (parameters of AOS and not DE) have been used in the experiments. The details on *irace* can be found in Chapter 4. The first eight baselines involve the DE algorithm with the following parameter values: population size ($NP = 100$), scaling factor ($F = 0.5$) and crossover rate ($CR = 1.0$). This choice for parameter F has shown good results [46]. CR as 1.0 has been chosen to see the full potential of mutation strategies to evolve each dimension of each parent. The results of LR and IPOP are taken from their original papers from the CEC2005 competition for the comparison.

B.1.1 Training and testing

We train three models of DE-DDQN on CEC2005 benchmark suite [157]. From the 25 functions of the benchmark suite, we excluded non-deterministic functions and functions without bounds (functions $F4$, $F7$, $F17$ and $F25$). The remaining 21 functions can be divided into four classes: unimodal functions $F1 – F5$; basic multimodal functions $F6 – F12$; expanded multimodal functions $F13 – F14$; and hybrid composition functions $F15 – F24$. We split these 21 functions into roughly 75% training and 25% testing sets, that is, 16 functions ($F1$, $F2$, $F5$, $F6$, $F8$, $F10$–$F15$, $F19$–$F22$ and $F24$) are assigned to the training set and the rest ($F3$, $F9$, $F16$, $F18$ and $F23$) are assigned to the test set. According to the above classification, the training set contains at least two functions from each class and the test set contains at least one function from each class except for expanded multimodal functions, as both functions of this class are included in the training set. For each function, we consider both dimensions 10 and 30, giving a total of 32 problems for training and 10 problems for testing.

During training, we cycle through the 32 training problems multiple times and keep track of the mean reward achieved in each cycle. The primary neural network weights are over-written if the mean reward is better than what we have observed in previous cycles. Once a cycle is finished, the 32 problems are shuffled before being used again. The mean reward stopped improving after 1890 cycles (60480 problems, 6048×10^5 FEs) which indicated the convergence of the learning process.

Once the training is over, NN weights are returned and used for the testing phase.[1] For testing, each DE-DDQN variant was independently run 25 times on each test problem and each run was stopped when either absolute error difference from the optimum is smaller than 10^{-8} or 10^4 function evaluations are exhausted. Mean and standard deviation of the final error values achieved by each of the 25 runs are reported in Table B.2.

TABLE B.2: Mean (and standard deviation in parenthesis) of function error values obtained by 25 runs for each function on test set. Former five are dimension 10 and last five are dimension 30. We refer DE-DDQN as DDQN. Bold entry is the minimum mean error found by any method for each function.

Function	Random	DE1	DE2	DE3	DE4	AdapSS	FAUC	RecPM	LR	IPOP	DDQN1	DDQN2	DDQN3
*F*3-10	2.34e+8	2.78e+8	2.26e+8	2.38e+8	2.63e+8	3.37e+4	3.53e+5	3.08e+4	**4.94e-9**	5.60e-9	3.98e+3	7.38 e+0	2.12e+1
	(1.06e+8)	(1.30e+8)	(1.10e+8)	(1.23e+8)	(1.42e+8)	(3.62e+5)	(1.65e+4)	(2.64e+4)	(1.45e-9)	(1.93e-9)	(1.91e+3)	(3.59e0)	(1.14e+1)
*F*9-10	1.20e+2	1.18e+2	1.22e+2	1.16e+2	1.22e+2	4.10e+1	4.36e+1	3.79e+1	8.60e+1	**6.21e+0**	4.19e+1	3.68e+1	3.86e+1
	(1.32e+1)	(1.20e+1)	(1.88e+1)	(1.44e+1)	(1.71e+1)	(6.36e+0)	(5.99e+0)	(6.33e+0)	(3.84e+1)	(2.10e+0)	(6.21e+0)	(4.64e+0)	(7.66e+0)
*F*16-10	6.46e+2	6.50e+2	6.31e+2	5.91e+2	6.33e+2	1.90e+2	2.05e+2	1.89e+2	1.49e+2	**1.11e+2**	1.93e+2	1.79e+2	1.88e+2
	(1.02e+2)	(9.65e+1)	(1.15e+2)	(1.07e+2)	(9.97e+1)	(2.21e+1)	(1.41e+1)	(1.25e+1)	(8.01e+1)	(1.66e+1)	(1.24e+1)	(2.05e+1)	(1.41e+1)
*F*18-10	1.33e+3	1.36e+3	1.39e+3	1.36e+3	1.36e+3	6.13e+2	6.94e+2	6.48e+2	8.40e+2	6.02e+2	**5.20e+2**	5.81e+2	5.98e+2
	(1.16e+2)	(8.81e+1)	(1.11e+2)	(1.09e+2)	(9.67e+1)	(1.67e+2)	(1.93e+2)	(1.82e+2)	(2.17e+2)	(2.76e+2)	(1.93e+2)	(2.47e+2)	(2.61e+2)
*F*23-10	1.49e+3	1.51e+3	1.51e+3	1.51e+3	1.49e+3	6.66e+2	7.73e+2	6.37e+2	1.22e+3	9.49e+2	**6.18e+2**	6.56e+2	6.90e+2
	(5.16e+1)	(6.71e+1)	(6.03e+1)	(5.58e+1)	(4.97e+1)	(1.99e+2)	(2.05e+2)	(1.23e+2)	(5.16e+2)	(3.52e+2)	(1.40e+2)	(1.57e+2)	(1.35e+2)
*F*3-30	2.48e+9	2.68e+9	2.50e+9	2.65e+9	2.51e+9	1.52e+7	6.44e+7	1.31e+7	**1.28e+6**	6.11e+6	1.52e+7	3.06e+6	5.72e+6
	(6.60e+8)	(7.84e+8)	(9.04e+8)	(6.69e+8)	(8.22e+8)	(5.50e+7)	(5.88e+6)	(6.84e+6)	(7.13e+5)	(3.79e+6)	(9.07e+6)	(2.54e+6)	(1.30e+7)
*F*9-30	5.33e+2	5.27e+2	5.42e+2	5.19e+2	5.41e+2	2.54e+2	2.88e+2	2.53e+2	4.19e+2	**4.78e+1**	2.73e+2	2.39e+2	2.73e+2
	(3.09e+1)	(3.40e+1)	(3.73e+1)	(4.53e+1)	(3.43e+1)	(2.69e+1)	(1.72e+1)	(1.26e+1)	(1.02e+2)	(1.15e+1)	(1.97e+1)	(1.52e+1)	(2.24e+1)
*F*16-30	1.19e+3	1.18e+3	1.18e+3	1.21e+3	1.20e+3	3.11e+2	3.48e+2	2.97e+2	2.52e+2	**1.96e+2**	3.18e+2	3.74e+2	3.39e+2
	(1.36e+2)	(1.72e+2)	(1.16e+2)	(1.35e+2)	(1.63e+2)	(6.26e+1)	(5.27e+1)	(3.00e+1)	(2.08e+2)	(1.45e+2)	(4.22e+1)	(9.03e+1)	(8.41e+1)
*F*18-30	1.41e+3	1.43e+3	1.41e+3	1.42e+3	1.42e+3	9.65e+2	1.02e+3	9.71e+2	9.64e+2	**9.08e+2**	1.04e+3	9.45e+2	9.48e+2
	(5.70e+1)	(4.70e+1)	(6.47e+1)	(4.59e+1)	(5.54e+1)	(5.59e+1)	(2.37e+1)	(2.31e+1)	(1.46e+2)	(2.76e+0)	(2.27e+1)	(1.42e+1)	(3.25e+1)
*F*23-30	1.58e+3	1.57e+3	1.55e+3	1.57e+3	1.57e+3	9.43e+2	1.10e+3	9.67e+2	7.51e+2	**6.92e+2**	1.17e+3	9.74e+2	9.64e+2
	(4.64e+1)	(4.05e+1)	(4.51e+1)	(4.14e+1)	(5.15e+1)	(1.40e+2)	(1.01e+2)	(1.30e+2)	(3.30e+2)	(2.38e+2)	(6.30e+1)	(1.69e+2)	(1.70e+2)

B.1.2 Discussion of results

The average rankings of each method among the 10 test problem instances are shown in Table B.3. The differences among the 13 algorithms are significant ($p < .01$) according to the non-parametric Friedman test. We conducted a post-hoc analysis using the best performing method (DE-DDQN2) among the newly proposed ones as

[1]The weights obtained after training are available on Github [149] together with the source code, and can be used for testing on similar functions including expanded multimodal.

TABLE B.3: Average ranking of all methods.

Algo	IPOP	DDQN2	DDQN3	RecPM	LR	AdapSS	DDQN1	FAUC	Random	DE3	DE2	DE4	DE1
Rank	2.3	3.3	4.1	4.4	4.4	4.9	5.4	7.2	10.5	10.8	10.8	11.4	11.5

TABLE B.3 (cont.): Average ranking of all methods.

the control method for pairwise comparisons with the other methods. The p-values adjusted for multiple comparisons [109] are shown in Table B.4. The differences between DE-DDQN2 and the five baselines, random selection of operators and single strategy DEs (DE1-DE4), are significant while differences with other methods are not. The analysis makes clear that the proposed method learns to adaptively select the strategy at different stages of a DE run.

While differences between the three reward definitions are not statistically significant, the rankings provide some evidence that R2 performs better than the other two definitions. R2 assigns fixed reward values and is invariant to differences in function ranges, whereas R1 and R3 involving raw functions values may mislead the NN when dealing with functions with different fitness ranges. Thus, R1 could not perform best on CEC2005 problem set due to inconsistent function ranges. Comparing with other methods proposed in the literature shows that DE variants with a suitable operator selection strategy can perform similarly to *CMA-ES* variants which are known to be the best performing methods for this class of problems.

To further analyze the difference between DE-DDQN and other AOS methods we provide boxplots of the results of 25 runs of DE-DDQN2, *PM-AdapSS* and *RecPM-AOS* on each function (Fig. B.1). We observe that the overall minimum function value found across the 25 runs is lower for DE-DDQN2 in all problems except *F*9-10 and *F*16-30. As seen in box plots, for *F*18 and *F*23 with dimension 10, DE-DDQN2 often gets stuck at local optima, but manages to find a better overall solution compared to the other methods. Other methods find high variance solutions in these cases. At the same time, the median values of solutions found are better for six out

TABLE B.4: Post-hoc (Li) using DE-DDQN2 as control method.

Comparison	**Statistic**	**Adjusted p-value**	**Result**
DDQN2 vs DE1	4.70819	0.00001	H0 is rejected
DDQN2 vs DE4	4.65077	0.00008	H0 is rejected
DDQN2 vs DE2	4.30627	0.00005	H0 is rejected
DDQN2 vs DE3	4.30627	0.00005	H0 is rejected
DDQN2 vs Random	4.13402	0.00010	H0 is rejected
DDQN2 vs *F-AUC-MAB*	2.23926	0.06630	H0 is not rejected
DDQN2 vs DDQN1	1.20576	0.39166	H0 is not rejected
DDQN2 vs *PM-AdapSS*	0.91867	0.50299	H0 is not rejected
DDQN2 vs *RecPM-AOS*	0.63159	0.59848	H0 is not rejected
DDQN2 vs LR	0.63159	0.59848	H0 is not rejected
DDQN2 vs IPOP	0.57417	0.61515	H0 is not rejected
DDQN2 vs DDQN3	0.45934	0.64599	H0 is not rejected

of ten problems. This observation suggests that incorporating restart strategies similar to those used by IPOP-CMAES can be particularly useful for DE-DDQN and give us a direction for future work. DE-DDQN2 performs well consistently for the unimodal $F3$ with both 10 and 30 dimensions, while the other AOS methods find relatively higher error solutions with high variance. We interpret this as an indication that DE-DDQN can identify this type of problem and apply a more suitable AOS strategy than Rec-PM and *PM-AdapSS*. On the other hand, we see that for $F16$-30 and $F23$-30, DE-DDQN2 exhibits higher variance of solutions, which suggests that higher dimensional multimodal functions often confuse the NN, leading it to suboptimal behaviour.

B.2 Conclusion

We presented results on CEC2005 benchmark set with DE-DDQN, a Deep-RL-based operator selection method. It is employed to online select the mutation strategies of DE. DE-DDQN has two phases, offline training and online evaluation phase. During training we collected data from DE runs using a reward metric to assess the performance of the selected mutation action and 99 features to evaluate the state of the DE. We employ three reward definitions, R1, R2 and R3 to reward the actions. The features and reward values are used to optimise the weights of a neural network to learn the most rewarding mutation given the DE state. The weights learned during training are then used during the online phase to predict the mutation strategy to use when solving a new problem. Experiments were run using 21 functions from CEC2005 benchmark suite, each function was evaluated with dimensions 10 and 30. A set of 32 functions was used for training and we run the online phase on a different test set of 10 functions. We compare the results of the three proposed DE-DDQN variants to several baseline DE algorithms using no online selection, random selection and other AOS methods, and also to the two winners of the CEC2005 competition.

All three proposed methods outperform all the non-AOS baselines based on mean error seen in 25 runs on test functions. This shows that the proposed methods

FIGURE B.1: Function error values obtained by 25 runs of DE-DDQN2, *RecPM-AOS* and *PM-AdapSS* for each function on test set with dimension 10 and 30

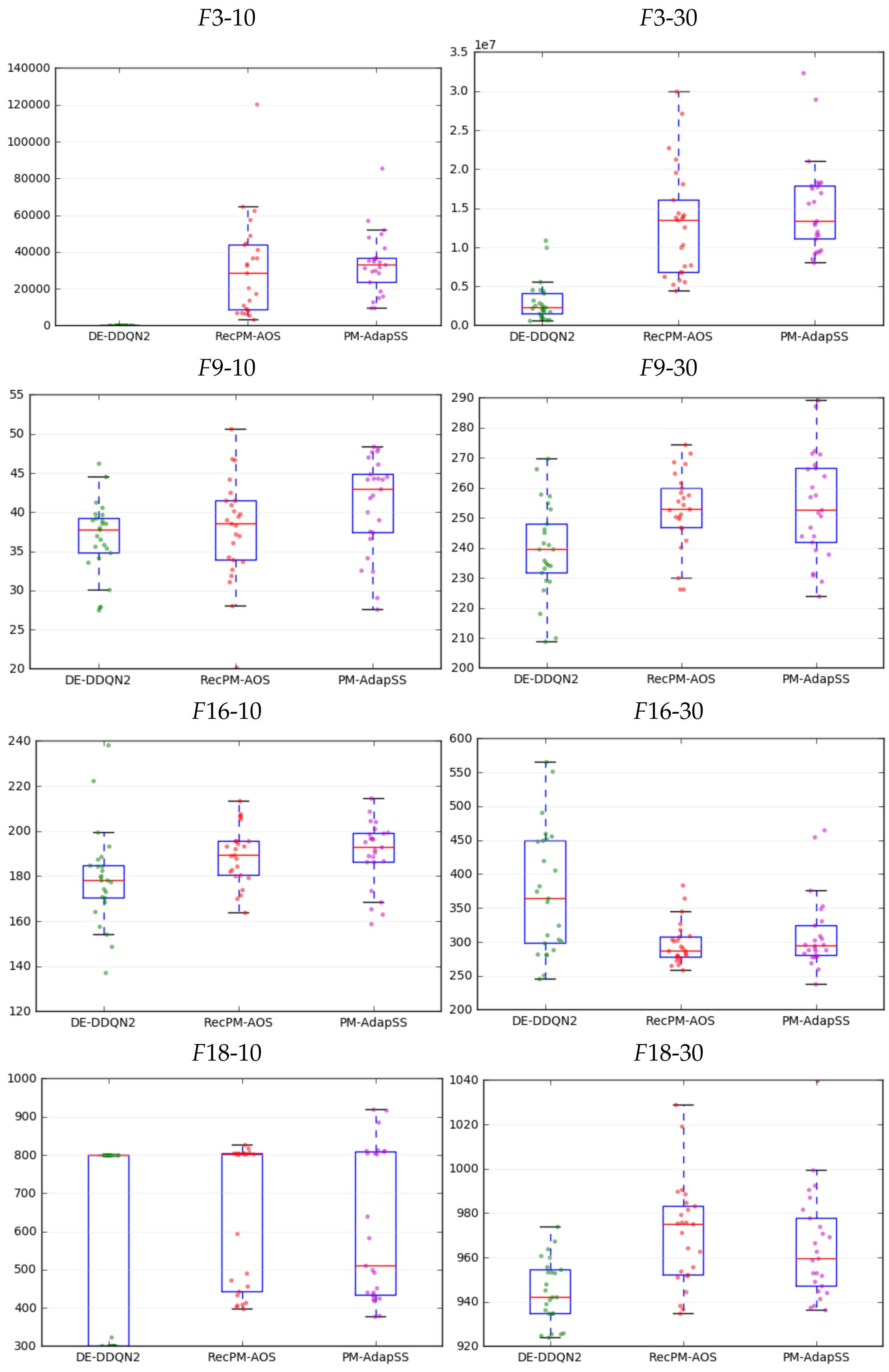
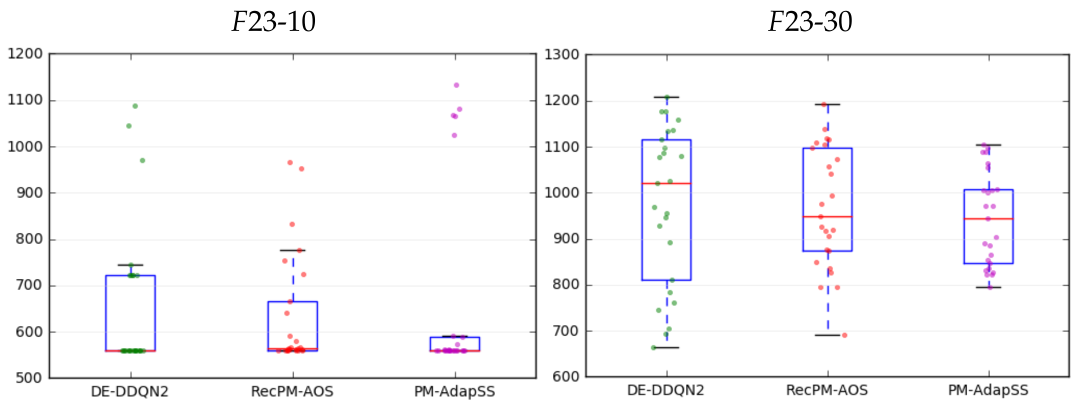


FIGURE B.1 (cont.): Function error values obtained by 25 runs of DE-DDQN2, *RecPM-AOS* and *PM-AdapSS* for each function on test set with dimension 10 and 30



can learn to select the right strategy at different stages of the algorithm. Our statistical analysis suggests that differences between the best proposed method and the AOS methods from the literature are not significant, but the best performing version of our model, DE-DDQN2, was ranked overall second after IPOP-CMAES. The R2 reward function, which assigns fixed reward values when better solutions are found, is more helpful for learning an AOS strategy.

Our experimental results show that the DE variants using AOS completely outperform the DE variants using a fixed mutation strategy or a random selection. Although a non-parametric post-hoc test does not find that the differences between the CMAES algorithms and the AOS-enabled DE algorithms (including DE-DDQN) are statistically significant, DE-DDQN is the second best approach, behind IPOP-CMAES, in terms of mean rank.

Appendix C

Average run time for various algorithms

Tables C.1 to C.6 show the aRT (average Run Time) in number of function evaluations obtained by various algorithms. aRT is calculated as the ratio of the number of function evaluations for reaching the target value over successful runs, plus the maximum number of evaluations for unsuccessful trials, divided by the number of successful trials, on 24 functions. The run time for a function becomes undefined if there are no successful runs. The obtained aRT is divided by the respective best aRT measured during BBOB-2009 in dimension 20. This aRT ratio and, in braces as dispersion measure, the half difference between 10 and 90%-tile of bootstrapped run lengths appear for each algorithm and target, the corresponding reference aRT in the first row. The different target Δf-values are shown in the top row. #succ is the number of trials that reached the (final) target $f_{\mathrm{opt}} + 10^{-8}$. The median number of conducted function evaluations is additionally given in *italics*, if the target in the last column was never reached. Best results are printed in bold. Entries, succeeded by a star, are statistically significantly better (according to the rank-sum test) when compared to all other algorithms of the table, with $p = 0.05$ or $p = 10^{-k}$ when the number k following the star is larger than 1, with Bonferroni correction by the number of functions (24). A $\downarrow$ indicates the same tested against the best algorithm from BBOB 2009. aRT shown for algorithm in Tables C.1 and C.2 are result of online selection of four operators while other are obtained by adaptation of nine operators.

TABLE C.1: Average runtime aRT

Δf_{opt}	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f1	43	43	43	43	43	43	43	15/15
JaDE	48(7)	94(7)	144(9)	193(7)	241(8)	341(9)	438(12)	13/13
PM-AdapSS3	101(20)	197(10)	292(6)	379(25)	467(17)	648(26)	831(20)	13/13
CMA-ES	96(21)	187(31)	279(44)	370(53)	461(69)	640(84)	821(106)	13/13
F-AUC3	118(42)	223(63)	321(30)	412(54)	512(101)	698(58)	887(76)	13/13
RecPM-AOS1	128(38)	256(66)	430(194)	993(1513)	1324(2177)	1737(2205)	2225(2068)	12/13
RecPM-AOS2	**7.5**(2)$^{\star 3}$	**13**(2)$^{\star 3}$	**20**(2)$^{\star 3}$	**26**(2)$^{\star 3}$	**33**(2)$^{\star 3}$	**45**(3)$^{\star 3}$	**58**(2)$^{\star 3}$	13/13
RecPM-AOS3	93(18)	180(24)	264(13)	348(26)	429(25)	595(36)	760(30)	13/13
f2	385	386	387	388	390	391	393	15/15
JaDE	**28**(1)$^{\star 3}$	**33**(1)$^{\star 3}$	**39**(1)$^{\star 3}$	**44**(1)*	49(2)	61(2)	71(3)	13/13
PM-AdapSS3	52(1)	63(3)	73(4)	83(2)	93(2)	113(2)	132(3)	13/13
CMA-ES	47(6)	57(8)	67(10)	77(8)	86(13)	105(12)	123(15)	13/13
F-AUC3	72(6)	90(15)	109(45)	128(34)	148(20)	190(27)	228(131)	13/13
RecPM-AOS1	63(15)	108(39)	185(18)	212(217)	295(256)	353(216)	380(163)	13/13
RecPM-AOS2	37(4)	44(3)	45(2)	47(2)	**47**(2)*	**48**(2)$^{\star 3}$	**50**(1)$^{\star 3}$	13/13
RecPM-AOS3	48(4)	58(3)	68(7)	77(6)	86(6)	105(5)	123(4)	13/13
f3	5066	7626	7635	7637	7643	7646	7651	15/15
JaDE	**6.4**(0.2)$^{\star 3}$	**6.0**(0.2)$^{\star 3}$	**6.8**(0.2)$^{\star 3}$	**7.6**(0.2)$^{\star 3}$	**8.3**(0.2)$^{\star 3}$	**10**(0.2)$^{\star 3}$	**11**(0.1)$^{\star 3}$	13/13
PM-AdapSS3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
CMA-ES	526(515)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC3	5023(3948)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS1	2497(2270)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS2	776(684)	∞	∞	∞	∞	∞	∞ *1e6*	0/13
RecPM-AOS3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
f4	4722	7628	7666	7686	7700	7758	1.4e5	9/15
JaDE	**8.1**(0.2)$^{\star 3}$	**7.0**(0.1)$^{\star 3}$	**8.0**(0.2)$^{\star 3}$	**8.9**(0.1)$^{\star 3}$	**10**(0.2)$^{\star 3}$	**11**(0.1)$^{\star 3}$	**0.71**(8e-3)$^{\star 3}$	13/13
PM-AdapSS3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
CMA-ES	5283(5718)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS1	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS2	∞	∞	∞	∞	∞	∞	∞ *1e6*	0/13
RecPM-AOS3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
f5	41	41	41	41	41	41	41	15/15
JaDE	42(6)	52(9)	53(6)	53(8)	53(8)	53(10)	53(6)	13/13
PM-AdapSS3	79(17)	89(16)	93(9)	93(18)	93(15)	93(20)	93(19)	13/13
CMA-ES	77(25)	92(29)	94(23)	95(20)	95(25)	95(23)	95(24)	13/13
F-AUC3	80(25)	91(24)	95(29)	95(23)	95(33)	95(31)	95(32)	13/13
RecPM-AOS1	106(64)	130(97)	136(69)	136(89)	136(86)	136(56)	136(119)	13/13
RecPM-AOS2	**5.1**(1)$^{\star 3}$	**6.1**(1)$^{\star 3}$	**6.2**(0.9)$^{\star 3}$	**6.2**(1)$^{\star 3}$	**6.2**(0.7)$^{\star 3}$	**6.2**(0.7)$^{\star 3}$	**6.2**(0.6)$^{\star 3}$	13/13
RecPM-AOS3	43(7)	52(9)	54(11)	54(8)	54(11)	54(15)	54(10)	13/13
f6	1296	2343	3413	4255	5220	6728	8409	15/15
JaDE	9.4(0.6)	7.8(0.8)	7.3(0.6)	7.3(0.8)	7.2(0.8)	7.4(0.9)	7.4(1)	13/13
PM-AdapSS3	20(2)	16(1.0)	15(1)	14(1)	14(0.8)	14(0.6)	14(0.5)	13/13
CMA-ES	17(1)	15(1)	14(1)	14(1)	14(1)	14(0.7)	14(0.8)	13/13
F-AUC3	32(3)	44(6)	90(178)	111(235)	141(259)	163(133)	186(322)	9/13
RecPM-AOS1	67(56)	53(49)	45(42)	43(23)	40(27)	40(20)	57(11)	12/13
RecPM-AOS2	**1.7**(0.4)$^{\star 3}$	**1.3**(0.3)$^{\star 3}$	**1.2**(0.2)$^{\star 3}$	**1.2**(0.1)$^{\star 3}$	**1.1**(0.1)$^{\star 3}$	**1.2**(0.1)$^{\star 3}$	**1.2**(0.1)$^{\star 3}$	13/13
RecPM-AOS3	19(2)	15(1)	14(0.9)	14(0.8)	14(0.6)	14(0.4)	14(0.4)	13/13
f7	1351	4274	9503	16523	16524	16524	16969	15/15
JaDE	4.9(0.7)	379(527)	1265(1263)	739(1195)	739(575)	739(1089)	719(1326)	1/13
PM-AdapSS3	5.9(0.7)	3.4(0.2)	2.1(0.1)	1.6(0.1)	1.6(0.1)	1.6(0.1)	1.7(0.1)	13/13
CMA-ES	6.2(0.7)	3.9(0.7)	2.4(0.4)	1.9(0.2)	1.9(0.4)	1.9(0.4)	1.9(0.2)	13/13
F-AUC3	7.7(2)	4.6(1)	2.9(0.6)	2.3(0.6)	2.3(0.6)	2.3(0.6)	2.3(0.5)	13/13
RecPM-AOS1	7.3(2)	4.4(0.8)	20(53)	12(61)	12(31)	12(0.6)	12(89)	12/13
RecPM-AOS2	**1.3**(0.1)$^{\star 3}$	208(114)	649(472)	∞	∞	∞	∞ *1e6*	0/13
RecPM-AOS3	5.2(0.5)	**3.1**(0.5)	**1.9**(0.2)	**1.6**(0.2)	**1.6**(0.1)	**1.6**(0.2)	**1.6**(0.2)	13/13
f8	2039	3871	4040	4148	4219	4371	4484	15/15
JaDE	18(1)	15(0.6)	16(0.4)	17(0.5)	17(0.5)	17(0.4)	18(0.7)	13/13
PM-AdapSS3	35(8)	35(4)	37(5)	38(5)	39(5)	40(4)	40(3)	13/13
CMA-ES	43(14)	47(17)	51(19)	52(19)	53(19)	54(18)	54(18)	13/13
F-AUC3	249(762)	246(299)	391(867)	385(350)	384(593)	391(437)	449(725)	7/13
RecPM-AOS1	52(34)	82(21)	82(19)	81(12)	81(127)	82(125)	82(122)	12/13
RecPM-AOS2	**3.8**(0.7)$^{\star 3}$	**4.3**(3)$^{\star 3}$	**4.7**(1)$^{\star 3}$	**4.8**(0.5)$^{\star 3}$	**4.8**(0.6)$^{\star 3}$	**4.9**(0.7)$^{\star 3}$	**4.9**(0.8)$^{\star 3}$	13/13
RecPM-AOS3	23(5)	21(2)	22(3)	23(3)	24(4)	25(2)	26(3)	13/13
f9	1716	3102	3277	3379	3455	3594	3727	15/15
JaDE	35(4)	30(3)	32(2)	32(1)	32(3)	33(2)	33(3)	13/13
PM-AdapSS3	37(9)	40(8)	43(7)	45(5)	45(6)	46(4)	46(4)	13/13
CMA-ES	51(23)	64(29)	71(36)	74(40)	76(41)	76(45)	76(42)	13/13
F-AUC3	246(250)	716(1307)	732(607)	741(361)	914(1319)	906(561)	1096(1111)	5/13
RecPM-AOS1	69(55)	52(11)	53(30)	53(29)	54(14)	55(11)	56(13)	13/13
RecPM-AOS2	**4.5**(1)$^{\star 3}$	**5.0**(0.5)$^{\star 3}$	**5.4**(0.7)$^{\star 3}$	**5.5**(0.2)$^{\star 3}$	**5.5**(0.3)$^{\star 3}$	**5.5**(0.7)$^{\star 3}$	**5.5**(0.3)$^{\star 3}$	13/13
RecPM-AOS3	26(4)	25(3)	27(2)	28(2)	29(2)	30(2)	31(2)	13/13
f10	7413	8661	10735	13641	14920	17073	17476	15/15
JaDE	12(6)	14(5)	15(5)	14(4)	14(4)	15(4)	18(3)	13/13
PM-AdapSS3	2.7(0.1)	2.8(0.1)	2.6(0.2)	2.4(0.1)	2.4(0.1)	2.6(0.1)	3.0(0.1)	13/13
CMA-ES	3.6(0.4)	4.1(0.9)	4.1(2)	4.0(2)	4.1(2)	4.7(0.8)	5.7(3)	13/13
F-AUC3	3.8(1)	5.1(0.4)	5.5(7)	5.5(8)	6.1(0.5)	6.2(9)	6.8(10)	13/13
RecPM-AOS1	3.3(0.8)	4.0(0.9)	4.3(1)	4.2(3)	8.2(3)	8.6(2)	10(4)	13/13
RecPM-AOS2	**1.7**(0.2)$^{\star 3}$	**1.7**(0.1)$^{\star 3}$	**1.6**(0.2)$^{\star 3}$	**1.3**(0.1)$^{\star 3}$	**1.2**(0.0)$^{\star 3}$	**1.1**(0.1)$^{\star 3}$	**1.1**(0.0)$^{\star 3}$	13/13
RecPM-AOS3	2.5(0.1)	2.5(0.2)	2.4(0.2)	2.1(0.1)	2.2(0.1)	2.4(0.1)	2.8(0.1)	13/13
f11	1002	2228	6278	8586	9762	12285	14831	15/15
JaDE	105(14)	52(117)	21(2)	17(2)	17(2)	16(22)	16(2)	12/13
PM-AdapSS3	8.7(0.8)	5.7(0.4)	2.6(0.2)	2.4(0.1)	2.5(0.1)	2.6(0.1)	2.7(0.1)	13/13
CMA-ES	10(0.7)	7.1(0.8)	3.5(0.4)	3.2(0.4)	3.4(0.6)	3.7(0.4)	4.4(2)	13/13
F-AUC3	8.9(2)	6.1(2)	2.9(0.7)	2.7(0.7)	2.8(0.7)	3.0(0.2)	3.1(0.8)	13/13
RecPM-AOS1	10(1)	7.1(1)	4.6(4)	6.0(0.8)	6.2(1)	6.3(3)	7.0(5)	13/13
RecPM-AOS2	11(0.9)	5.3(0.1)	**2.0**(0.1)$^{\star 3}$	**1.5**(0.1)$^{\star 3}$	**1.4**(0.0)$^{\star 3}$	**1.2**(0.0)$^{\star 3}$	**1.0**(0.0)$^{\star 3}$	13/13
RecPM-AOS3	**7.3**(1)	**5.0**(0.5)	2.3(0.2)	2.1(0.3)	2.3(0.2)	2.4(0.2)	2.5(0.1)	13/13
f12	1042	1938	2740	3156	4140	12407	13827	15/15
JaDE	19(4)	20(11)	28(10)	31(10)	29(9)	13(3)	14(3)	13/13
PM-AdapSS3	29(2)	18(0.9)	20(1)	23(10)	23(10)	12(4)	13(3)	13/13
CMA-ES	32(2)	22(16)	24(10)	31(11)	32(8)	17(8)	19(3)	13/13
F-AUC3	35(10)	22(6)	27(8)	41(23)	273(618)	129(403)	129(181)	7/13
RecPM-AOS1	62(69)	50(44)	57(41)	60(47)	50(34)	20(10)	20(10)	13/13
RecPM-AOS2	**2.0**(0.1)$^{\star 3}$	**3.0**(2)$^{\star 3}$	**3.8**(2)$^{\star 3}$	**4.2**(2)$^{\star 3}$	**3.8**(2)$^{\star 3}$	**1.7**(0.7)$^{\star 3}$	**1.8**(0.6)$^{\star 3}$	13/13
RecPM-AOS3	27(6)	20(12)	20(7)	21(10)	20(5)	9.5(2)	10(2)	13/13

TABLE C.2: Average runtime aRT

Δf_{opt}	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f13	652	2021	2751	3507	18749	24455	30201	15/15
JaDE	17(0.7)	14(5)	15(5)	14(4)	3.7(0.7)	4.8(1.0)	8.7(1)	13/13
PM-AdapSS3	28(1)	13(0.6)	12(0.2)	12(0.3)	2.6(0.1)	2.6(0.1)	2.6(0.1)	13/13
CMA-ES	29(5)	14(1)	14(2)	14(1)	3.2(0.4)	3.6(0.8)	4.1(1)	13/13
F-AUC3	36(7)	17(4)	16(3)	15(4)	3.5(0.8)	4.1(2)	6.7(13)	13/13
RecPM-AOS1	45(62)	34(26)	35(19)	31(18)	6.6(4)	8.1(4)	7.3(2)	13/13
RecPM-AOS2	**6.1**(6)$^{\star 3}$	**5.2**(2)$^{\star 2}$	**4.6**(2)$^{\star 3}$	**4.5**(3)$^{\star 3}$	**2.1**(3)	5.3(5)	9.4(9)	10/13
RecPM-AOS3	25(2)	12(1)	11(0.5)	11(0.7)	2.5(0.1)	**2.5**(0.1)	**2.5**(0.2)	13/13
f14	75	239	304	451	932	1648	15661	15/15
JaDE	18(6)	18(1)	23(2)	25(2)	20(2)	36(11)	68(119)	4/13
PM-AdapSS3	43(7)	34(2)	43(1)	40(1)	25(2)	20(3)	2.8(0.1)	13/13
CMA-ES	40(7)	34(5)	42(6)	40(4)	26(4)	24(3)	4.3(2)	13/13
F-AUC3	53(14)	45(8)	59(13)	53(11)	32(7)	26(5)	4.3(2)	13/13
RecPM-AOS1	45(19)	42(11)	52(14)	49(11)	35(21)	39(22)	5.3(4)	13/13
RecPM-AOS2	**4.1**(1.0)$^{\star 3}$	**3.0**(0.5)$^{\star 3}$	**3.6**(0.3)$^{\star 3}$	**4.3**(0.6)$^{\star 3}$	**4.2**(0.5)$^{\star 3}$	**6.2**(0.5)$^{\star 3}$	**1.2**(0.1)$^{\star 3}$	13/13
RecPM-AOS3	33(4)	30(2)	38(5)	36(4)	23(2)	20(2)	2.8(0.3)	13/13
f15	30378	1.5e5	3.1e5	3.2e5	3.2e5	4.5e5	4.6e5	15/15
JaDE	**36**(7)	∞	∞	∞	∞	∞	∞ *1e6*	0/13
PM-AdapSS3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
CMA-ES	202(116)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC3	413(1021)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS1	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS2	37(41)	∞	∞	∞	∞	∞	∞ *1e6*	0/13
RecPM-AOS3	408(200)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
f16	**1384**	**27265**	**77015**	**1.4e5**	**1.9e5**	**2.0e5**	**2.2e5**	15/15
JaDE	24(5)	∞	∞	∞	∞	∞	∞ *1e6*	0/13
PM-AdapSS3	1.9e4(1e4)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
CMA-ES	4288(5552)	466(439)	**335**(214)	∞	∞	∞	∞ *2e6*	0/13
F-AUC3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS1	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS2	**1.9**(0.6)$^{\star 3}$	**3.0**(1)$^{\star 3}$	∞	∞	∞	∞	∞ *1e6*	0/13
RecPM-AOS3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
f17	63	1030	4005	12242	30677	56288	80472	15/15
JaDE	8.0(3)	7.3(3)	4.4(0.8)	**2.5**(0.3)	**1.7**(0.3)	8.1(15)	19(21)	2/13
PM-AdapSS3	23(13)	13(0.7)	6.6(0.4)	3.4(0.4)	1.9(0.2)	1.8(0.2)	17(19)	7/13
CMA-ES	17(20)	12(1)	6.1(0.6)	3.2(0.3)	1.8(0.2)	**1.8**(0.2)	6.1(12)	11/13
F-AUC3	35(12)	16(4)	8.3(2)	4.3(0.7)	2.3(0.5)	2.2(0.3)	**4.0**(0.3)	10/13
RecPM-AOS1	24(14)	15(4)	8.1(2)	4.2(1)	14(17)	8.7(27)	13(31)	9/13
RecPM-AOS2	**2.7**(2)	**0.94**(0.4)$^{\star 3}$	**1.4**(3)	3.8(7)	8.8(5)	∞	∞ *1e6*	0/13
RecPM-AOS3	22(14)	13(3)	6.6(0.5)	3.3(0.3)	1.9(0.1)	1.9(0.2)	6.2(19)	11/13
f18	621	3972	19561	28555	67569	1.3e5	1.5e5	15/15
JaDE	7.3(1)	4.5(1)	**1.6**(0.4)	9.4(6)	20(16)	∞	∞ *1e6*	0/13
PM-AdapSS3	13(2)	5.3(0.5)	1.8(0.2)	1.9(0.3)	6.4(0.0)	**5.5**(8)	**7.1**(14)	7/13
CMA-ES	10(2)	4.7(0.8)	1.8(0.2)	1.9(0.2)	3.6(0.2)	7.8(11)	7.1(7)	9/13
F-AUC3	15(5)	6.2(1)	2.2(0.5)	2.2(0.3)	3.7(0.2)	14(27)	32(24)	4/13
RecPM-AOS1	14(3)	6.2(1)	2.2(0.5)	2.4(0.7)	3.9(0.2)	14(19)	17(10)	5/13
RecPM-AOS2	**1.2**(0.3)$^{\star 3}$	**2.0**(2)	4.6(6)	423(411)	∞	∞	∞ *1e6*	0/13
RecPM-AOS3	11(2)	4.6(0.7)	1.7(0.2)	**1.8**(0.2)	**3.5**(0.2)	14(12)	47(38)	3/13
f19	**1**	**1**	**3.4e5**	**4.7e6**	**6.2e6**	**6.7e6**	**6.7e6**	15/15
JaDE	845(267)	7.7e5(4e5)	∞	∞	∞	∞	∞ *1e6*	0/13
PM-AdapSS3	1891(355)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
CMA-ES	**1**(0)	**1**(0)	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC3	**1**(0)	**1**(0)	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS1	**1**(0)	**1**(0)	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS2	156(69)	4.9e4(4e4)	∞	∞	∞	∞	∞ *1e6*	0/13
RecPM-AOS3	1305(322)	8.2e6(9e6)	∞	∞	∞	∞	∞ *2e6*	0/13
f20	82	46150	3.1e6	5.5e6	5.5e6	5.6e6	5.6e6	14/15
JaDE	24(3)	**1.2**(0.2)$^{\star 3}$	**0.46**(0.6)$^{\star 3}$	**0.74**(1)$^{\star 3}$	**1.1**(2)$^{\star 3}$	∞$^{\star 3}$	∞ *1e6*$^{\star 3}$	0/13
PM-AdapSS3	47(7)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
CMA-ES	50(10)	32(7)	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC3	59(15)	556(466)	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS1	68(12)	556(769)	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS2	**5.1**(2)$^{\star 3}$	134(195)	∞	∞	∞	∞	∞ *1e6*	0/13
RecPM-AOS3	41(4)	557(856)	∞	∞	∞	∞	∞ *2e6*	0/13
f21	561	6541	14103	14318	14643	15567	17589	15/15
JaDE	7.4(3)	26(51)	16(11)	16(24)	15(11)	15(24)	14(15)	12/13
PM-AdapSS3	12(3)	264(306)	474(425)	467(1013)	456(683)	430(385)	380(426)	3/13
CMA-ES	309(4)	358(459)	320(425)	315(349)	308(444)	290(321)	257(370)	4/13
F-AUC3	18(16)	359(536)	228(319)	225(524)	220(546)	207(161)	184(228)	5/13
RecPM-AOS1	18(6)	690(612)	320(248)	315(908)	308(307)	290(321)	257(427)	4/13
RecPM-AOS2	**5.5**(9)	**11**(9)	**6.0**(5)	**5.9**(3)	**5.8**(7)	**5.5**(3)	**4.9**(3)	13/13
RecPM-AOS3	12(2)	690(459)	781(957)	769(524)	752(1263)	708(2023)	627(426)	2/13
f22	467	5580	23491	24163	24948	26847	1.3e5	12/15
JaDE	28(81)	246(462)	∞	∞	∞	∞	∞ *1e6*	0/13
PM-AdapSS3	1918(2143)	4303(2778)	∞	∞	∞	∞	∞ *2e6*	0/13
CMA-ES	15(5)	576(896)	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC3	376(6)	809(1075)	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS1	19(6)	809(986)	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS2	**4.2**(8)*	**48**(101)	**120**(93)	**117**(90)	**113**(162)	**105**(112)	**21**(26)	4/13
RecPM-AOS3	795(1075)	1973(1344)	∞	∞	∞	∞	∞ *2e6*	0/13
f23	**3.0**	**1614**	**67457**	**3.7e5**	**4.9e5**	**8.1e5**	**8.4e5**	15/15
JaDE	2.1(2)	**123**(61)*	∞	∞	∞	∞	∞ *1e6*	0/13
PM-AdapSS3	**1.6**(2)	7253(1e4)	∞	∞	∞	∞	∞ *2e6*	0/13
CMA-ES	2.1(2)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC3	3.0(2)	3782(3996)	378(422)	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS1	2.7(3)	5295(4966)	192(148)	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS2	3.4(3)	531(613)	**54**(45)	∞	∞	∞	∞ *1e6*	0/13
RecPM-AOS3	1.7(2)	2954(3952)	381(311)	∞	∞	∞	∞ *2e6*	0/13
f24	**1.3e6**	**7.5e6**	**5.2e7**	**5.2e7**	**5.2e7**	**5.2e7**	**5.2e7**	3/15
JaDE	∞	∞	∞	∞	∞	∞	∞ *1e6*	0/13
PM-AdapSS3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
CMA-ES	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS1	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS2	∞	∞	∞	∞	∞	∞	∞ *1e6*	0/13
RecPM-AOS3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13

TABLE C.3: Average runtime aRT

Δf_{opt}	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f1	43	43	43	43	43	43	43	15/15
U-AOS-FW	73(11)	141(8)	217(11)	294(17)	368(18)	527(21)	670(35)	13/13
R-SHADE	**28**(7)^{*3}	**61**(10)^{*3}	**91**(11)^{*3}	**122**(14)^{*3}	**152**(13)^{*3}	**211**(17)^{*3}	**269**(11)^{*3}	13/13
RecPM-AOS	86(16)	169(14)	255(29)	345(15)	434(37)	611(38)	790(56)	13/13
PM-AdapSS	92(14)	192(16)	297(14)	400(27)	507(16)	710(17)	926(16)	13/13
F-AUC-MAB	58(7)	113(6)	164(9)	219(16)	270(32)	376(24)	480(45)	13/13
Compass	67(9)	141(10)	217(12)	291(17)	368(10)	517(12)	667(13)	13/13
JaDE	48(10)	94(9)	144(10)	193(8)	241(9)	341(10)	438(12)	13/13
f2	385	386	387	388	390	391	393	15/15
U-AOS-FW	40(3)	48(2)	57(4)	66(4)	74(5)	91(5)	107(5)	13/13
R-SHADE	**18**(2)	**21**(2)	**24**(2)	**28**(2)	**31**(2)	**37**(3)	**43**(3)	13/13
RecPM-AOS	45(5)	55(4)	65(3)	74(5)	83(4)	103(6)	122(6)	13/13
PM-AdapSS	55(3)	67(0.9)	78(3)	90(3)	101(3)	124(2)	147(2)	13/13
F-AUC-MAB	24(2)	30(2)	35(7)	40(7)	46(5)	56(15)	67(8)	13/13
Compass	39(1)	47(2)	55(2)	64(2)	71(2)	88(2)	104(2)	13/13
JaDE	28(1)	33(1)	39(2)	44(2)	49(2)	61(3)	71(3)	13/13
f3	5066	7626	7635	7637	7643	7646	7651	15/15
U-AOS-FW	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
R-SHADE	7.6(0.5)	7.1(0.3)	7.6(0.3)	7.9(0.2)	**8.1**(0.2)	**8.4**(0.2)^{*3}	**8.7**(0.3)^{*3}	13/13
RecPM-AOS	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
PM-AdapSS	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC-MAB	11(0.8)	8.4(0.9)	8.7(0.4)	9.0(0.8)	9.3(0.6)	10(1)	11(1)	13/13
Compass	5069(4441)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
JaDE	**6.4**(0.3)^{*3}	**6.0**(0.2)^{*3}	**6.8**(0.2)^{*3}	**7.6**(0.2)*	8.3(0.1)	10(0.1)	11(0.1)	13/13
f4	4722	7628	7666	7686	7700	7758	1.4e5	9/15
U-AOS-FW	2563(3709)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
R-SHADE	9.5(0.8)	8.3(0.4)	11(5)	12(0.4)	12(11)	12(11)	**0.70**(0.3)	13/13
RecPM-AOS	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
PM-AdapSS	5416(3812)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC-MAB	12(2)	125(0.7)	313(587)	313(520)	312(260)	311(451)	17(25)	6/13
Compass	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
JaDE	**8.1**(0.3)^{*2}	**7.0**(0.1)^{*3}	**8.0**(0.2)^{*3}	**8.9**(0.2)	**10**(0.1)	**11**(0.2)	0.71(9e-3)	13/13
f5	41	41	41	41	41	41	41	15/15
U-AOS-FW	47(8)	55(7)	56(12)	57(7)	57(9)	57(9)	57(11)	13/13
R-SHADE	114(11)	210(19)	307(25)	399(30)	488(19)	676(34)	860(45)	13/13
RecPM-AOS	**33**(4)	**37**(8)	**39**(7)^{*2}	**39**(9)^{*2}	**39**(7)^{*2}	**39**(9)^{*2}	**39**(8)^{*2}	13/13
PM-AdapSS	69(9)	83(11)	85(10)	85(8)	85(22)	85(13)	85(6)	13/13
F-AUC-MAB	49(3)	59(9)	62(11)	63(10)	63(8)	63(8)	63(7)	13/13
Compass	42(7)	50(6)	51(12)	51(11)	51(7)	51(10)	51(7)	13/13
JaDE	42(6)	52(8)	53(7)	53(3)	53(5)	53(6)	53(7)	13/13
f6	1296	2343	3413	4255	5220	6728	8409	15/15
U-AOS-FW	14(2)	12(1)	11(0.6)	11(1)	11(1.0)	11(0.5)	12(0.3)	13/13
R-SHADE	**4.1**(0.5)^{*3}	**3.9**(0.4)^{*3}	**3.8**(0.5)^{*3}	**3.9**(0.3)^{*3}	**3.9**(0.5)^{*3}	**4.0**(0.4)^{*3}	**4.1**(0.4)^{*3}	13/13
RecPM-AOS	19(2)	15(2)	14(1)	14(1)	14(0.9)	14(0.9)	14(0.8)	13/13
PM-AdapSS	18(2)	15(1)	14(0.9)	14(0.6)	14(0.4)	15(0.4)	15(0.2)	13/13
F-AUC-MAB	23(4)	23(5)	23(9)	26(8)	29(15)	39(24)	53(33)	13/13
Compass	14(1)	11(0.8)	10(0.7)	11(0.5)	10(0.5)	11(0.4)	11(0.1)	13/13
JaDE	9.4(0.8)	7.8(0.6)	7.3(0.8)	7.3(0.9)	7.2(0.9)	7.4(0.4)	7.4(0.6)	13/13
f7	1351	4274	9503	16523	16524	16524	16969	15/15
U-AOS-FW	5.2(0.6)	3.7(0.6)	2.3(0.2)	1.9(0.1)	1.9(0.1)	1.9(0.2)	1.9(0.2)	13/13
R-SHADE	**2.1**(0.3)^{*3}	33(19)	76(87)	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS	6.5(1.0)	4.6(0.6)	3.0(0.3)	2.5(0.3)	2.5(0.3)	2.5(0.4)	2.6(0.4)	13/13
PM-AdapSS	6.7(0.5)	43(117)	20(0.5)	12(0.2)	12(31)	12(0.2)	12(88)	12/13
F-AUC-MAB	47(29)	1025(1075)	∞	∞	∞	∞	∞ *2e6*	0/13
Compass	5.1(0.6)	**3.5**(0.7)	**2.2**(0.4)	**1.8**(0.2)	**1.8**(0.2)	**1.8**(0.2)	**1.9**(0.2)	13/13
JaDE	4.9(1.0)	379(761)	1265(2157)	739(726)	739(1044)	739(1044)	719(368)	1/13
f8	2039	3871	4040	4148	4219	4371	4484	15/15
U-AOS-FW	22(2)	22(2)	23(2)	24(2)	24(2)	25(2)	27(1)	13/13
R-SHADE	**10**(0.9)^{*3}	**14**(8)	**15**(7)	**15**(7)	**15**(7)	**16**(3)	**16**(7)	13/13
RecPM-AOS	19(4)	59(259)	59(1)	58(121)	58(1)	59(1)	60(1)	12/13
PM-AdapSS	31(5)	33(2)	36(2)	36(1)	37(2)	38(2)	39(2)	13/13
F-AUC-MAB	461(414)	3238(9429)	∞	∞	∞	∞	∞ *2e6*	0/13
Compass	20(2)	19(1)	20(2)	21(1)	22(1)	23(1)	24(0.8)	13/13
JaDE	18(0.7)	15(0.5)	16(0.4)	17(0.6)	17(0.4)	17(0.5)	18(0.7)	13/13
f9	1716	3102	3277	3379	3455	3594	3727	15/15
U-AOS-FW	31(1)	34(3)	37(3)	38(3)	39(3)	42(3)	44(3)	13/13
R-SHADE	**16**(4)^{*3}	**20**(2)^{*2}	**22**(13)^{*2}	**23**(3)^{*2}	**24**(2)^{*2}	**24**(2)^{*2}	**24**(2)^{*2}	13/13
RecPM-AOS	25(1)	25(1)	28(1)	30(1)	31(1)	35(2)	39(2)	13/13
PM-AdapSS	44(3)	51(4)	55(3)	56(5)	57(3)	58(5)	60(5)	13/13
F-AUC-MAB	689(775)	8342(7253)	∞	∞	∞	∞	∞ *2e6*	0/13
Compass	27(2)	145(323)	141(154)	139(3)	138(290)	136(280)	134(2)	11/13
JaDE	35(3)	30(3)	32(2)	32(2)	32(1)	33(2)	33(3)	13/13
f10	7413	8661	10735	13641	14920	17073	17476	15/15
U-AOS-FW	6.4(0.7)	7.1(1)	6.8(1)	6.4(2)	6.5(1)	7.0(0.3)	8.0(2)	13/13
R-SHADE	16(6)	22(6)	25(6)	26(6)	28(7)	32(8)	40(14)	9/13
RecPM-AOS	8.0(0.7)	8.5(0.5)	8.3(0.7)	7.5(0.4)	7.9(0.5)	8.6(0.5)	10(0.4)	13/13
PM-AdapSS	5.6(0.5)	6.3(1)	**6.0**(1)	**5.5**(2)	**5.7**(0.8)	**6.1**(1)	**7.0**(2)	13/13
F-AUC-MAB	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
Compass	**5.5**(0.9)	**6.3**(2)	6.2(1)	6.0(1)	6.3(1)	7.1(3)	8.3(3)	13/13
JaDE	12(4)	14(7)	15(4)	14(4)	14(3)	15(3)	18(5)	13/13
f11	1002	2228	6278	8586	9762	12285	14831	15/15
U-AOS-FW	15(2)	11(0.8)	5.7(0.3)	5.4(0.3)	5.7(0.1)	6.2(0.2)	6.5(0.3)	13/13
R-SHADE	**7.9**(5)*	13(5)	8.0(3)	8.2(2)	10(2)	11(3)	12(4)	13/13
RecPM-AOS	23(3)	16(2)	8.0(0.4)	7.4(0.2)	7.9(0.6)	8.6(0.4)	9.0(0.6)	13/13
PM-AdapSS	15(1)	10(0.4)	5.2(0.2)	4.9(0.2)	5.2(0.2)	5.6(0.2)	5.8(0.2)	13/13
F-AUC-MAB	917(159)	942(459)	4071(8920)	∞	∞	∞	∞ *2e6*	0/13
Compass	13(2)	**10**(0.9)	**4.8**(0.3)*	**4.4**(0.3)^{*2}	**4.8**(0.4)*	**5.1**(0.6)*	**5.4**(0.5)^{*2}	13/13
JaDE	105(501)	52(117)	21(2)	17(31)	17(28)	16(41)	16(3)	12/13
f12	1042	1938	2740	3156	4140	12407	13827	15/15
U-AOS-FW	29(2)	21(4)	25(12)	32(13)	33(8)	17(4)	19(3)	13/13
R-SHADE	**8.5**(1)^{*3}	**17**(11)	**22**(13)	**25**(12)	**24**(9)	**12**(4)	**13**(4)	13/13
RecPM-AOS	37(5)	31(19)	31(20)	37(15)	37(8)	18(3)	20(3)	13/13
PM-AdapSS	44(18)	40(10)	42(31)	47(29)	44(13)	20(2)	22(3)	13/13
F-AUC-MAB	1091(2590)	1500(653)	4257(6447)	∞	∞	∞	∞ *2e6*	0/13
Compass	29(2)	31(21)	38(19)	41(19)	38(12)	18(8)	19(6)	13/13
JaDE	19(2)	20(14)	28(13)	31(10)	29(8)	13(3)	14(3)	13/13

TABLE C.4: Average runtime aRT

Δf_{opt}	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f13	652	2021	2751	3507	18749	24455	30201	15/15
U-AOS-FW	25(3)	13(2)	15(1)	17(2)	4.2(0.5)	5.0(0.3)	5.6(0.5)	13/13
R-SHADE	**12**(7)	**8.4**(3)	**11**(2)	**12**(3)	**3.5**(1)	5.2(1)	27(22)	0/13
RecPM-AOS	31(4)	17(1)	19(2)	22(2)	5.5(0.2)	6.5(0.3)	7.2(0.5)	13/13
PM-AdapSS	32(1)	16(1)	18(1)	19(0.6)	4.4(0.2)	4.9(0.1)	**5.2**(0.1)	13/13
F-AUC-MAB	49(8)	504(1134)	2877(2212)	∞	∞	∞	∞ *2e6*	0/13
Compass	24(2)	13(0.6)	14(0.6)	15(0.6)	3.8(0.4)	**4.5**(0.5)	5.7(1)	13/13
JaDE	17(1)	14(6)	15(5)	14(4)	3.7(0.3)	4.8(0.9)	8.7(2)	13/13
f14	75	239	304	451	932	1648	15661	15/15
U-AOS-FW	29(10)	26(1)	35(0.9)	35(1)	26(0.7)	29(2)	5.3(0.9)	12/13
R-SHADE	**8.1**(2)^{*2}	**10**(2)^{*3}	**13**(2)^{*3}	**13**(2)^{*3}	**11**(2)^{*3}	62(18)	1606(2171)	0/13
RecPM-AOS	32(4)	31(4)	40(4)	41(3)	31(1)	39(2)	6.5(0.5)	13/13
PM-AdapSS	33(9)	33(3)	45(1.0)	44(2)	30(2)	31(1)	**4.9**(0.2)	13/13
F-AUC-MAB	37(9)	33(3)	40(5)	91(26)	2.6e4(5e4)	∞	∞ *2e6*	0/13
Compass	23(5)	24(2)	32(2)	32(1)	23(1)	**26**(1.0)	7.4(0.3)	11/13
JaDE	18(8)	18(2)	23(2)	25(2)	20(0.9)	36(26)	68(88)	4/13
f15	30378	1.5e5	3.1e5	3.2e5	3.2e5	4.5e5	4.6e5	15/15
U-AOS-FW	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
R-SHADE	56(35)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
PM-AdapSS	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC-MAB	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
Compass	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
JaDE	**36**(22)	∞	∞	∞	∞	∞	∞ *1e6*	0/13
f16	1384	27265	77015	1.4e5	1.9e5	2.0e5	2.2e5	15/15
U-AOS-FW	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
R-SHADE	27(22)	**280**(290)	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
PM-AdapSS	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC-MAB	944(431)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
Compass	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
JaDE	**24**(6)	∞	∞	∞	∞	∞	∞ *1e6*	0/13
f17	63	1030	4005	12242	30677	56288	80472	15/15
U-AOS-FW	14(4)	9.4(0.7)	5.2(0.4)	2.8(0.2)	1.7(0.3)	1.7(0.1)	3.6(6)	12/13
R-SHADE	**4.0**(2)	**3.4**(0.5)^{*3}	**4.1**(12)*	14(27)	51(57)	∞	∞ *2e6*	0/13
RecPM-AOS	16(11)	11(1)	6.9(0.7)	3.7(0.3)	2.1(0.2)	2.2(0.1)	2.0(0.3)	12/13
PM-AdapSS	16(9)	11(2)	6.5(0.4)	3.5(0.3)	2.0(0.1)	1.9(0.1)	6.2(12)	11/13
F-AUC-MAB	13(5)	26(14)	66(124)	182(544)	788(962)	∞	∞ *2e6*	0/13
Compass	15(12)	9.1(1)	5.1(0.6)	2.8(0.3)	**1.6**(0.1)	**1.6**(0.1)	**1.4**(0.1)	12/13
JaDE	8.0(6)	7.3(1)	4.4(0.7)	**2.5**(0.4)	1.7(2)	8.1(8)	19(20)	2/13
f18	621	3972	19561	28555	67569	1.3e5	1.5e5	15/15
U-AOS-FW	8.7(2)	4.6(0.6)	1.9(0.4)	2.1(0.3)	**1.3**(0.1)	3.9(8)	3.8(3)	11/13
R-SHADE	**3.1**(0.7)^{*3}	**2.1**(1.0)^{*3}	34(32)	910(788)	∞	∞	∞ *2e6*	0/13
RecPM-AOS	11(2)	5.9(0.6)	2.6(0.2)	3.0(0.5)	1.8(0.2)	**1.7**(0.2)	**1.9**(0.2)	12/13
PM-AdapSS	10(0.8)	5.1(0.6)	2.2(0.3)	2.5(1)	6.7(15)	4.0(0.1)	10(14)	8/13
F-AUC-MAB	20(6)	188(54)	∞	∞	∞	∞	∞ *2e6*	0/13
Compass	8.5(1)	4.5(0.5)	1.8(0.2)	**2.1**(0.2)	3.7(7)	3.9(4)	5.3(7)	10/13
JaDE	7.3(1)	4.5(1)	**1.6**(0.4)	9.4(13)	20(21)	∞	∞ *1e6*	0/13
f19	1	1	3.4e5	4.7e6	6.2e6	6.7e6	6.7e6	15/15
U-AOS-FW	**1**(0)	**1**(0)	∞	∞	∞	∞	∞ *2e6*	0/13
R-SHADE	344(84)	1.1e6(2e6)	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS	**1**(0)	**1**(0)	∞	∞	∞	∞	∞ *2e6*	0/13
PM-AdapSS	**1**(0)	**1**(0)	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC-MAB	**1**(0)	**1**(0)	∞	∞	∞	∞	∞ *2e6*	0/13
Compass	**1**(0)	**1**(0)	∞	∞	∞	∞	∞ *2e6*	0/13
JaDE	845(175)	7.7e5(4e5)	∞	∞	∞	∞	∞ *1e6*	0/13
f20	82	46150	3.1e6	5.5e6	5.5e6	5.6e6	5.6e6	14/15
U-AOS-FW	36(4)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
R-SHADE	**11**(3)^{*3}	1.3(0.2)	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS	44(7)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
PM-AdapSS	46(4)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC-MAB	36(6)	2.3(1)	**0.16**(0.2)*	**0.10**(0.0)^{*2}	**0.10**(0.1)^{*2}	**0.10**(8e-3)^{*2}	**0.10**(0.2)^{*2}	12/13
Compass	35(3)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
JaDE	24(3)	**1.2**(0.2)	0.46(0.3)	0.74(0.7)	1.1(1)	∞	∞ *1e6*	0/13
f21	561	6541	14103	14318	14643	15567	17589	15/15
U-AOS-FW	8.1(2)	263(229)	122(106)	120(210)	118(68)	111(161)	99(171)	7/13
R-SHADE	**3.0**(1)^{*3}	**6.5**(9)	**7.0**(6)	**7.0**(4)	**6.9**(4)	**6.6**(16)	**5.9**(14)	13/13
RecPM-AOS	659(893)	689(917)	320(355)	315(559)	308(307)	290(385)	257(341)	4/13
PM-AdapSS	309(6)	1683(2523)	781(390)	769(1257)	752(1639)	708(2280)	627(398)	2/13
F-AUC-MAB	32(36)	39(53)	80(109)	121(92)	223(149)	466(390)	659(279)	2/13
Compass	305(1783)	490(764)	320(284)	315(244)	308(273)	290(578)	257(227)	4/13
JaDE	7.4(2)	26(82)	16(15)	16(41)	15(4)	15(36)	14(23)	12/13
f22	**467**	**5580**	**23491**	**24163**	**24948**	**26847**	**1.3e5**	12/15
U-AOS-FW	1913(1072)	1196(806)	∞	∞	∞	∞	∞ *2e6*	0/13
R-SHADE	16(2)	**29**(33)	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS	791(2145)	1197(1792)	∞	∞	∞	∞	∞ *2e6*	0/13
PM-AdapSS	369(3)	808(1613)	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC-MAB	388(47)	216(327)	∞	∞	∞	∞	∞ *2e6*	0/13
Compass	**10**(4)	808(806)	∞	∞	∞	∞	∞ *2e6*	0/13
JaDE	28(65)	246(403)	∞	∞	∞	∞	∞ *1e6*	0/13
f23	3.0	1614	67457	3.7e5	4.9e5	8.1e5	8.4e5	15/15
U-AOS-FW	2.4(2)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
R-SHADE	2.5(2)	**88**(61)	**11**(12)^{*3}	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS	2.4(2)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
PM-AdapSS	**1.8**(3)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC-MAB	2.0(2)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
Compass	2.8(3)	∞	∞	∞	∞	∞	∞ *2e6*	0/13
JaDE	2.1(2)	123(31)	∞	∞	∞	∞	∞ *1e6*	0/13
f24	1.3e6	7.5e6	5.2e7	5.2e7	5.2e7	5.2e7	5.2e7	3/15
U-AOS-FW	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
R-SHADE	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
RecPM-AOS	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
PM-AdapSS	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
F-AUC-MAB	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
Compass	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/13
JaDE	∞	∞	∞	∞	∞	∞	∞ *1e6*	0/13

TABLE C.5: Average runtime aRT

Δf_{opt}	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f1	43	43	43	43	43	43	43	15/15
DE-DDQN1	270(0)	557(0)	849(0)	1143(0)	1391(0)	1997(0)	2605(0)	1/1
DE-DDQN2	90(0)	179(0)	308(0)	411(0)	517(0)	735(0)	982(0)	1/1
DE-DDQN3	**34**(0)	89(0)	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	47(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	47(0)	100(0)	149(0)	193(0)	244(0)	344(0)	431(0)	1/1
JaDE	66(0)	138(0)	205(0)	298(0)	368(0)	546(0)	677(0)	1/1
U-AOS-FW	42(0)	**68**(0)	**101**(0)	**133**(0)	**165**(0)	**230**(0)	**287**(0)	1/1
f2	385	386	387	388	390	391	393	15/15
DE-DDQN1	121(0)	154(0)	186(0)	219(0)	245(0)	312(0)	373(0)	1/1
DE-DDQN2	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	29(0)	34(0)	41(0)	45(0)	50(0)	62(0)	72(0)	1/1
JaDE	44(0)	52(0)	60(0)	69(0)	77(0)	96(0)	111(0)	1/1
U-AOS-FW	**16**(0)	**20**(0)	**23**(0)	**26**(0)	**29**(0)	**37**(0)	**44**(0)	1/1
f3	5066	7626	7635	7637	7643	7646	7651	15/15
DE-DDQN1	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN2	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	**6.3**(0)	**5.8**(0)	**6.7**(0)	**7.4**(0)	8.3(0)	10(0)	11(0)	1/1
JaDE	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
U-AOS-FW	7.7(0)	7.0(0)	7.5(0)	7.8(0)	**8.0**(0)	**8.4**(0)	**8.7**(0)	1/1
f4	4722	7628	7666	7686	7700	7758	1.4e5	9/15
DE-DDQN1	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN2	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	**7.5**(0)	**7.1**(0)	**8.0**(0)	**8.9**(0)	**10**(0)	11(0)	0.71(0)	1/1
JaDE	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
U-AOS-FW	10(0)	8.5(0)	9.2(0)	10(0)	10(0)	**10**(0)	**0.58**(0)	1/1
f5	41	41	41	41	41	41	41	15/15
DE-DDQN1	5693(0)	4.2e4(0)	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN2	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	**47**(0)	**51**(0)	**51**(0)	**51**(0)	**51**(0)	**51**(0)	**51**(0)	1/1
JaDE	50(0)	62(0)	62(0)	62(0)	62(0)	62(0)	62(0)	1/1
U-AOS-FW	139(0)	244(0)	329(0)	414(0)	503(0)	703(0)	862(0)	1/1
f6	1296	2343	3413	4255	5220	6728	8409	15/15
DE-DDQN1	37(0)	42(0)	46(0)	49(0)	53(0)	57(0)	60(0)	1/1
DE-DDQN2	18(0)	21(0)	19(0)	21(0)	24(0)	28(0)	30(0)	1/1
DE-DDQN3	**3.4**(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	10(0)	8.0(0)	7.4(0)	7.5(0)	7.3(0)	7.4(0)	7.3(0)	1/1
JaDE	14(0)	11(0)	11(0)	10(0)	11(0)	11(0)	11(0)	1/1
U-AOS-FW	3.8(0)	**3.6**(0)	**3.8**(0)	**3.9**(0)	**3.9**(0)	**4.1**(0)	**4.3**(0)	1/1
f7	1351	4274	9503	16523	16524	16524	16969	15/15
DE-DDQN1	47(0)	51(0)	25(0)	15(0)	15(0)	15(0)	15(0)	1/1
DE-DDQN2	3.8(0)	5.0(0)	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	3.7(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	6.9(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	4.0(0)	**3.4**(0)	**2.2**(0)	13(0)	13(0)	13(0)	12(0)	1/1
JaDE	5.4(0)	3.7(0)	2.4(0)	**1.8**(0)	**1.8**(0)	**1.8**(0)	**2.0**(0)	1/1
U-AOS-FW	**1.8**(0)	13(0)	20(0)	∞	∞	∞	∞ *2e6*	0/1
f8	2039	3871	4040	4148	4219	4371	4484	15/15
DE-DDQN1	199(0)	136(0)	148(0)	163(0)	178(0)	203(0)	231(0)	1/1
DE-DDQN2	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	19(0)	16(0)	17(0)	17(0)	17(0)	17(0)	18(0)	1/1
JaDE	21(0)	23(0)	24(0)	25(0)	25(0)	26(0)	27(0)	1/1
U-AOS-FW	**11**(0)	**11**(0)	**12**(0)	**12**(0)	**12**(0)	**13**(0)	**14**(0)	1/1
f9	1716	3102	3277	3379	3455	3594	3727	15/15
DE-DDQN1	84(0)	104(0)	109(0)	113(0)	115(0)	117(0)	119(0)	1/1
DE-DDQN2	15(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	33(0)	27(0)	29(0)	30(0)	30(0)	30(0)	30(0)	1/1
JaDE	30(0)	33(0)	36(0)	37(0)	38(0)	40(0)	42(0)	1/1
U-AOS-FW	**15**(0)	**19**(0)	**21**(0)	**22**(0)	**22**(0)	**23**(0)	**23**(0)	1/1
f10	7413	8661	10735	13641	14920	17073	17476	15/15
DE-DDQN1	7.7(0)	7.9(0)	7.5(0)	6.8(0)	7.1(0)	7.6(0)	8.8(0)	1/1
DE-DDQN2	**3.4**(0)	**3.7**(0)	**3.5**(0)	**3.2**(0)	**3.2**(0)	**3.5**(0)	**4.0**(0)	1/1
DE-DDQN3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	12(0)	12(0)	13(0)	14(0)	15(0)	17(0)	21(0)	1/1
JaDE	5.6(0)	6.2(0)	5.8(0)	5.3(0)	5.5(0)	6.0(0)	7.1(0)	1/1
U-AOS-FW	10(0)	20(0)	26(0)	23(0)	26(0)	33(0)	60(0)	0/1
f11	1002	2228	6278	8586	9762	12285	14831	15/15
DE-DDQN1	26(0)	17(0)	7.9(0)	7.4(0)	7.8(0)	8.0(0)	8.4(0)	1/1
DE-DDQN2	**6.7**(0)	**4.2**(0)	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	18(0)	11(0)	6.1(0)	**5.3**(0)	6.2(0)	7.8(0)	8.1(0)	1/1
JaDE	16(0)	12(0)	**5.8**(0)	5.6(0)	**5.8**(0)	**6.3**(0)	**6.6**(0)	1/1
U-AOS-FW	15(0)	24(0)	14(0)	12(0)	13(0)	15(0)	17(0)	1/1
f12	1042	1938	2740	3156	4140	12407	13827	15/15
DE-DDQN1	77(0)	50(0)	45(0)	52(0)	55(0)	27(0)	29(0)	1/1
DE-DDQN2	29(0)	**19**(0)	**16**(0)	**18**(0)	∞	∞	∞ *2e6*	0/1
DE-DDQN3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	16(0)	27(0)	35(0)	37(0)	34(0)	15(0)	**16**(0)	1/1
JaDE	26(0)	27(0)	32(0)	36(0)	36(0)	17(0)	18(0)	1/1
U-AOS-FW	**8.3**(0)	27(0)	34(0)	36(0)	**33**(0)	**15**(0)	16(0)	1/1

TABLE C.6: Average runtime aRT

Δf_{opt}	1e1	1e0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f13	652	2021	2751	3507	18749	24455	30201	15/15
DE-DDQN1	113(0)	97(0)	126(0)	138(0)	34(0)	38(0)	41(0)	1/1
DE-DDQN2	13(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	14(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	18(0)	18(0)	17(0)	17(0)	3.8(0)	**4.9**(0)	8.2(0)	1/1
JaDE	25(0)	14(0)	17(0)	17(0)	4.3(0)	5.2(0)	**5.6**(0)	1/1
U-AOS-FW	**11**(0)	**5.1**(0)	**5.1**(0)	**13**(0)	**2.8**(0)	5.2(0)	9.3(0)	0/1
f14	75	239	304	451	932	1648	15661	15/15
DE-DDQN1	61(0)	75(0)	108(0)	106(0)	68(0)	55(0)	7.5(0)	1/1
DE-DDQN2	21(0)	25(0)	45(0)	40(0)	26(0)	23(0)	**3.3**(0)	1/1
DE-DDQN3	16(0)	17(0)	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	26(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	23(0)	19(0)	24(0)	25(0)	20(0)	24(0)	36(0)	0/1
JaDE	27(0)	22(0)	35(0)	36(0)	27(0)	28(0)	4.6(0)	1/1
U-AOS-FW	**7.9**(0)	**8.9**(0)	**12**(0)	**12**(0)	**10**(0)	**20**(0)	∞ *2e6*	0/1
f15	30378	1.5e5	3.1e5	3.2e5	3.2e5	4.5e5	4.6e5	15/15
DE-DDQN1	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN2	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	**10**(0)	∞	∞	∞	∞	∞	∞ *1e6*	0/1
JaDE	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
U-AOS-FW	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
f16	1384	27265	77015	1.4e5	1.9e5	2.0e5	2.2e5	15/15
DE-DDQN1	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN2	144(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	122(0)	**7.8**(0)	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	**4.1**(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	24(0)	∞	∞	∞	∞	∞	∞ *1e6*	0/1
JaDE	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
U-AOS-FW	58(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
f17	63	1030	4005	12242	30677	56288	80472	15/15
DE-DDQN1	11(0)	20(0)	9.2(0)	4.2(0)	2.1(0)	1.7(0)	1.5(0)	1/1
DE-DDQN2	14(0)	5.0(0)	4.4(0)	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	7.1(0)	5.9(0)	3.7(0)	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	17(0)	14(0)	7.7(0)	8.7(0)	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	**3.2**(0)	7.0(0)	4.4(0)	2.3(0)	**1.4**(0)	3.4(0)	2.6(0)	0/1
JaDE	16(0)	9.2(0)	5.0(0)	2.7(0)	1.6(0)	**1.7**(0)	**1.5**(0)	1/1
U-AOS-FW	5.7(0)	**3.2**(0)	**2.0**(0)	**1.3**(0)	∞	∞	∞ *2e6*	0/1
f18	621	3972	19561	28555	67569	1.3e5	1.5e5	15/15
DE-DDQN1	22(0)	12(0)	4.7(0)	5.0(0)	2.8(0)	2.4(0)	2.7(0)	1/1
DE-DDQN2	15(0)	6.1(0)	2.9(0)	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	5.5(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	5.7(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	7.6(0)	4.2(0)	**1.4**(0)	15(0)	∞	∞	∞ *1e6*	0/1
JaDE	9.1(0)	4.2(0)	1.7(0)	**2.0**(0)	**1.3**(0)	**1.1**(0)	**1.3**(0)	1/1
U-AOS-FW	**3.1**(0)	**2.7**(0)	72(0)	∞	∞	∞	∞ *2e6*	0/1
f19	1	1	3.4e5	4.7e6	6.2e6	6.7e6	6.7e6	15/15
DE-DDQN1	**1**(0)	**1**(0)	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN2	**1**(0)	**1**(0)	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	**1**(0)	**1**(0)	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	**1**(0)	**1**(0)	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	771(0)	4.1e5(0)	∞	∞	∞	∞	∞ *1e6*	0/1
JaDE	**1**(0)	**1**(0)	∞	∞	∞	∞	∞ *2e6*	0/1
U-AOS-FW	333(0)	1.4e6(0)	∞	∞	∞	∞	∞ *2e6*	0/1
f20	82	46150	3.1e6	5.5e6	5.5e6	5.6e6	5.6e6	14/15
DE-DDQN1	99(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN2	33(0)	**0.85**(0)	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	27(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	20(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	21(0)	1.0(0)	**0.16**(0)	∞	∞	∞	∞ *1e6*	0/1
JaDE	40(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
U-AOS-FW	**10**(0)	1.4(0)	∞	∞	∞	∞	∞ *2e6*	0/1
f21	561	6541	14103	14318	14643	15567	17589	15/15
DE-DDQN1	29(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN2	5.0(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	2.9(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	6.8(0)	**1.0**(0)	**0.62**(0)	**0.73**(0)	**0.82**(0)	**0.96**(0)	**1.0**(0)	1/1
U-AOS-FW	9.2(0)	1.2(0)	0.64(0)	0.77(0)	0.89(0)	1.1(0)	1.2(0)	1/1
U-AOS-F	**2.8**(0)	3.3(0)	1.6(0)	1.6(0)	1.6(0)	1.7(0)	1.6(0)	1/1
f22	467	5580	23491	24163	24948	26847	1.3e5	12/15
DE-DDQN1	28(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN2	7.9(0)	**1.8**(0)	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	8.2(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	95(0)	∞	∞	∞	∞	∞	∞ *1e6*	0/1
JaDE	8.7(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
U-AOS-FW	**3.7**(0)	54(0)	∞	∞	∞	∞	∞ *2e6*	0/1
f23	3.0	1614	67457	3.7e5	4.9e5	8.1e5	8.4e5	15/15
DE-DDQN1	1(0)	382(0)	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN2	2.3(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	3.0(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	7.7(0)	835(0)	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	**0.67**(0)	117(0)	∞	∞	∞	∞	∞ *1e6*	0/1
JaDE	6.0(0)	∞	∞	∞	∞	∞	∞ *2e6*	0/1
U-AOS-FW	1.3(0)	**41**(0)	**12**(0)	∞	∞	∞	∞ *2e6*	0/1
f24	1.3e6	7.5e6	5.2e7	5.2e7	5.2e7	5.2e7	5.2e7	3/15
DE-DDQN1	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN2	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN3	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
DE-DDQN4	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
F-AUC-MAB	∞	∞	∞	∞	∞	∞	∞ *1e6*	0/1
JaDE	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1
U-AOS-FW	∞	∞	∞	∞	∞	∞	∞ *2e6*	0/1

Appendix D

Operator selection and best fitness graphs for DE-DDQN2, DE-DDQN3 and DE-DDQN4

FIGURE D.1: Operator application and best fitness graphs for DE-DDQN2. Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"

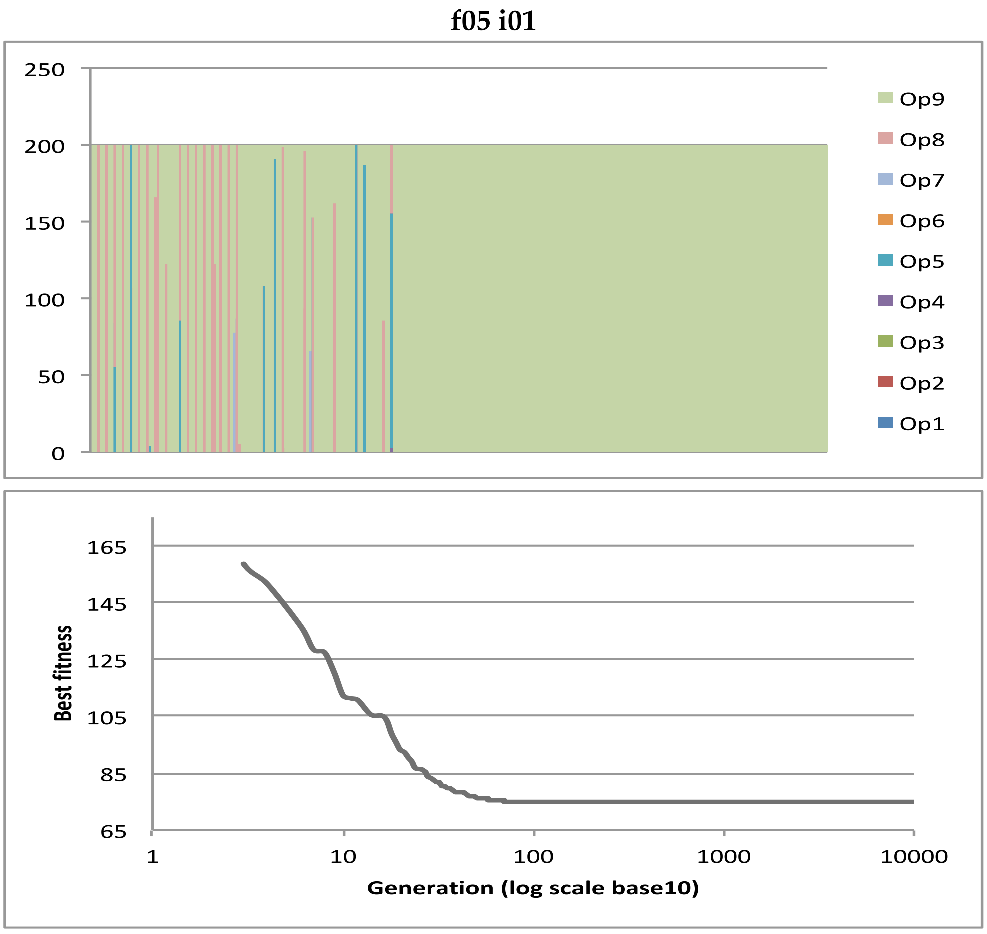
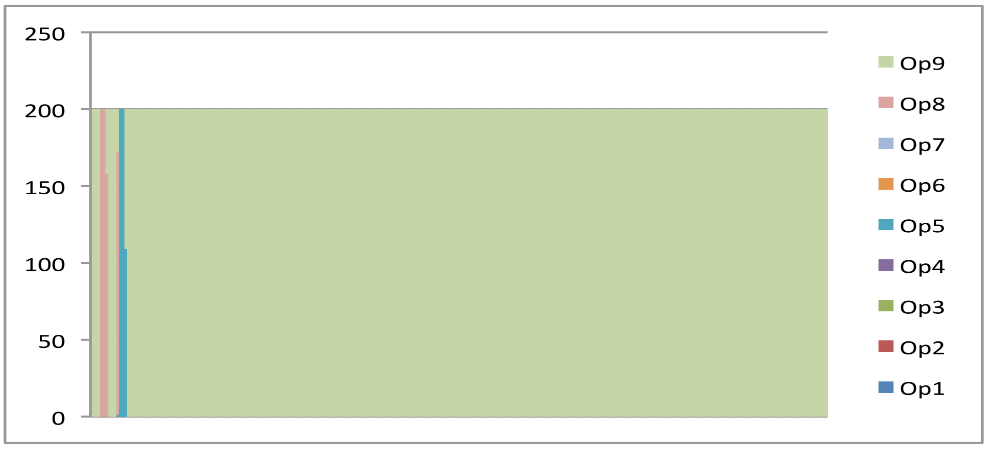
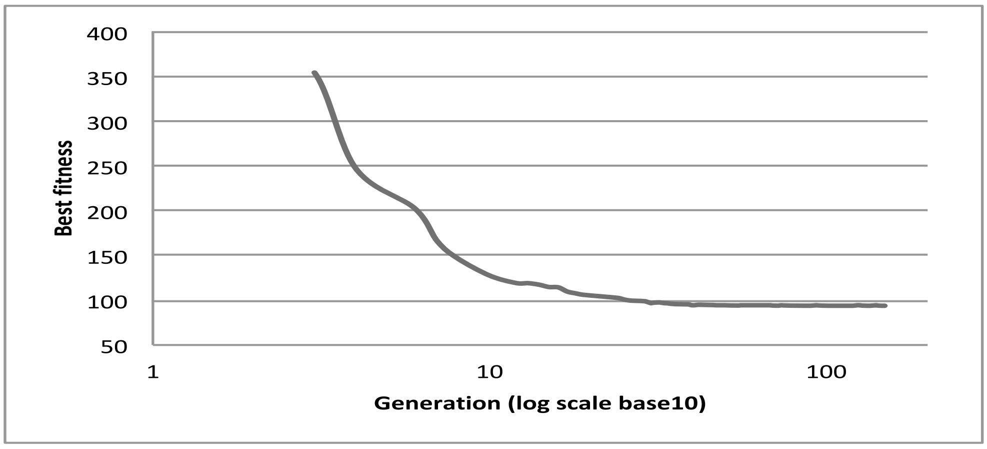


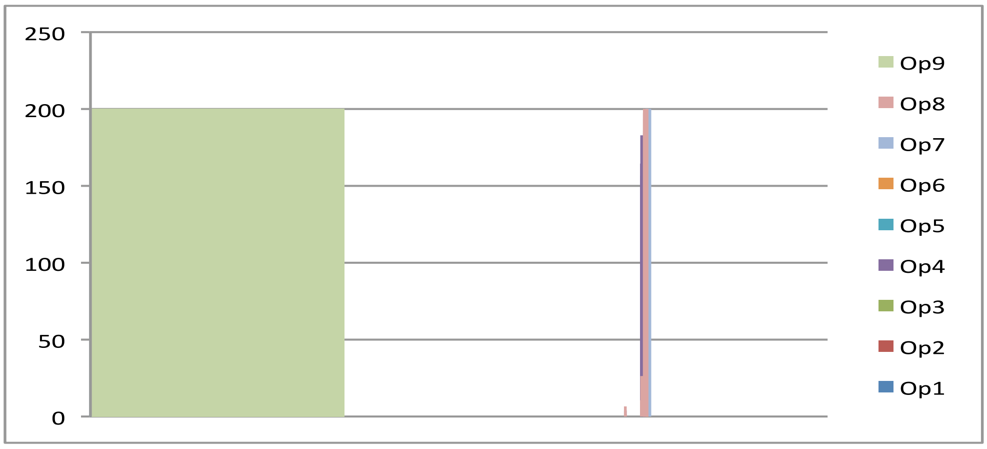
FIGURE D.1 (cont.): Operator application and best fitness graphs for DE-DDQN2. Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"

f07 i01





f04 i02



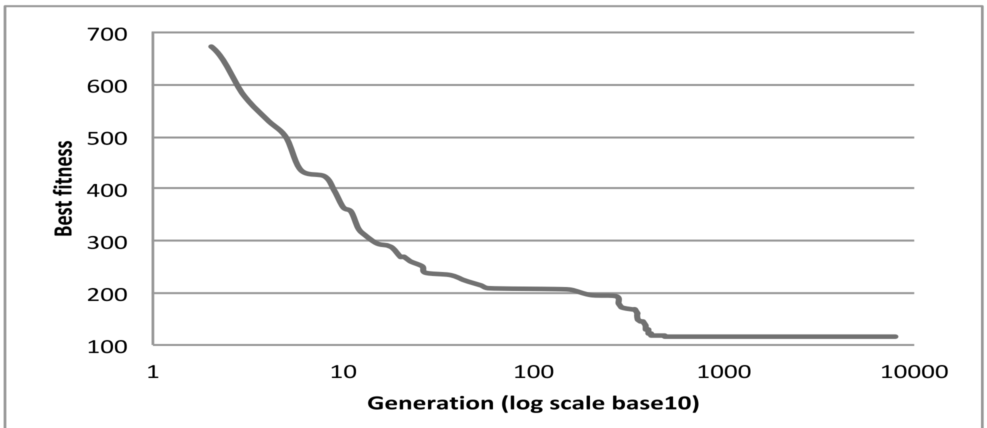
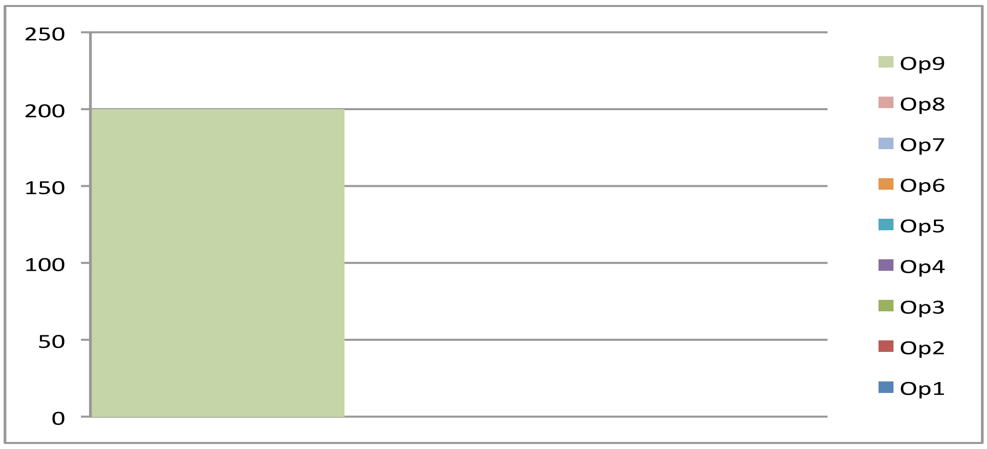
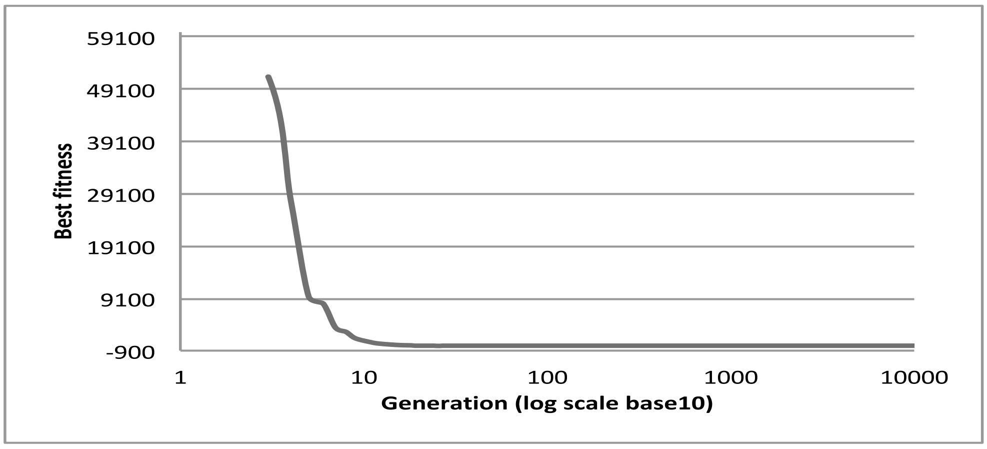


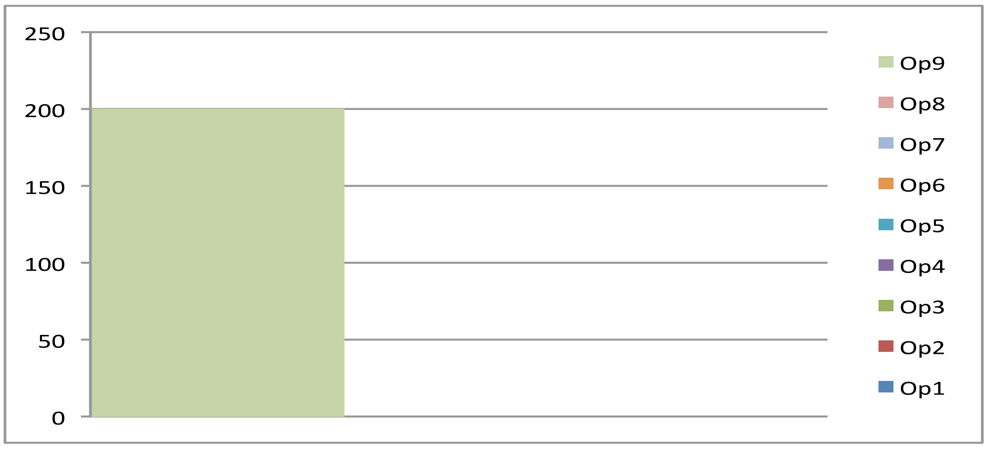
FIGURE D.1 (cont.): Operator application and best fitness graphs for DE-DDQN2. Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"

f08 i10





f013 i14



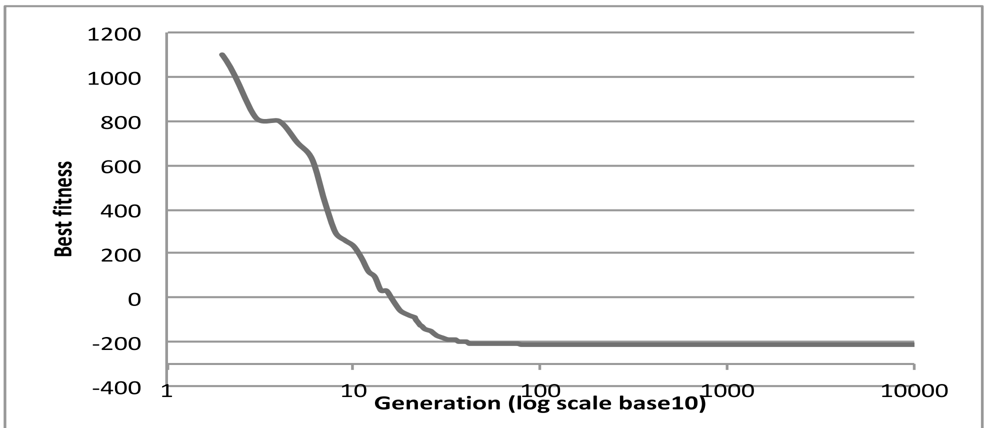
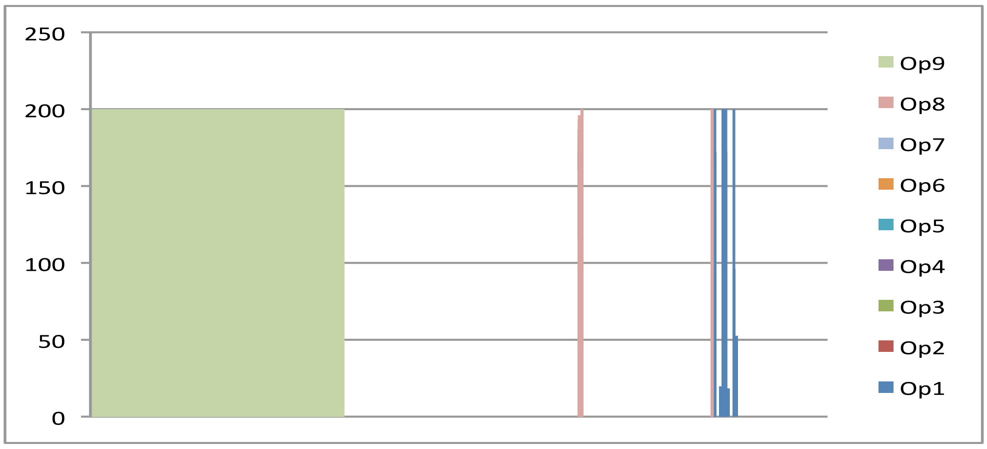
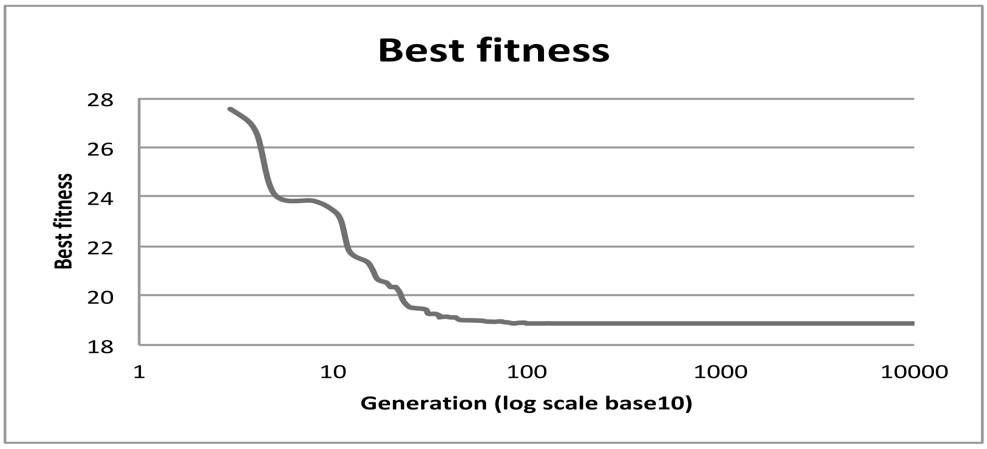


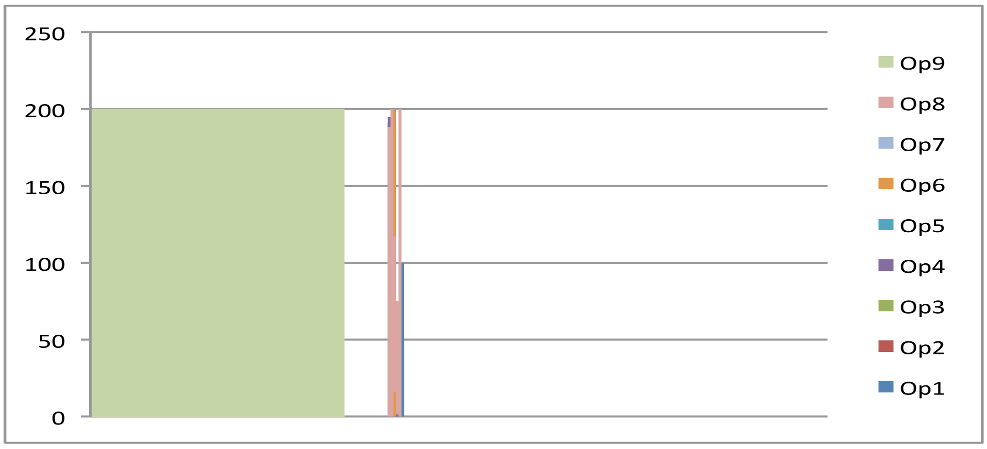
FIGURE D.1 (cont.): Operator application and best fitness graphs for DE-DDQN2. Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"

f17 i02





f23 i04



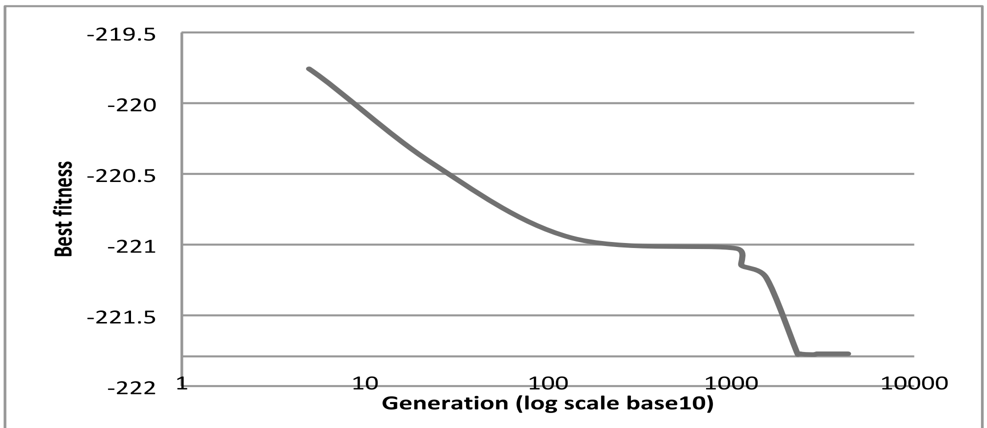
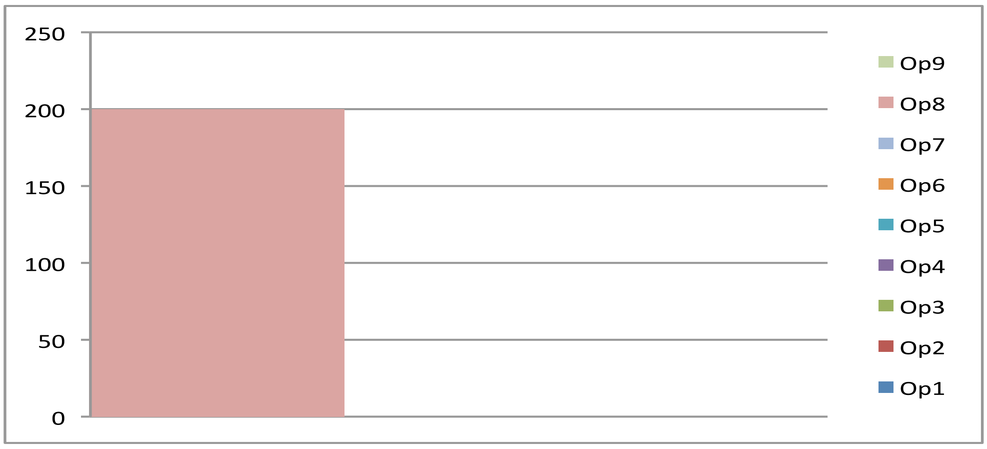
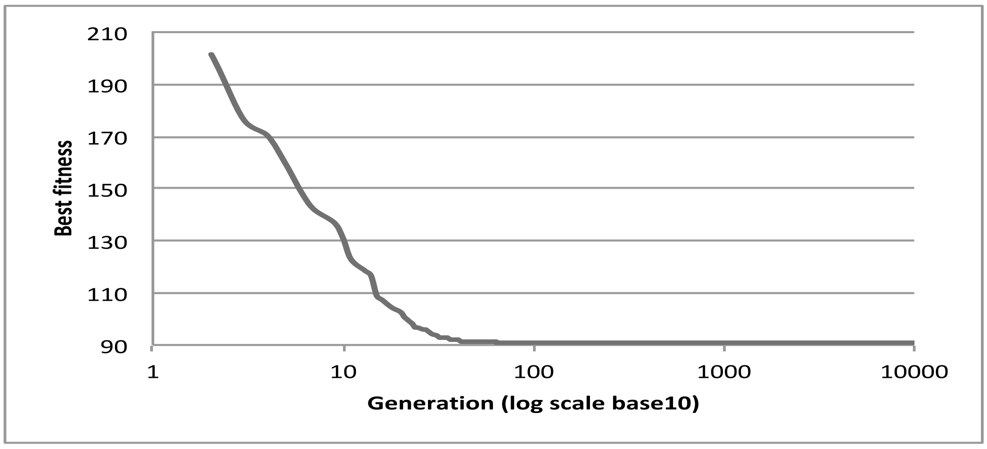


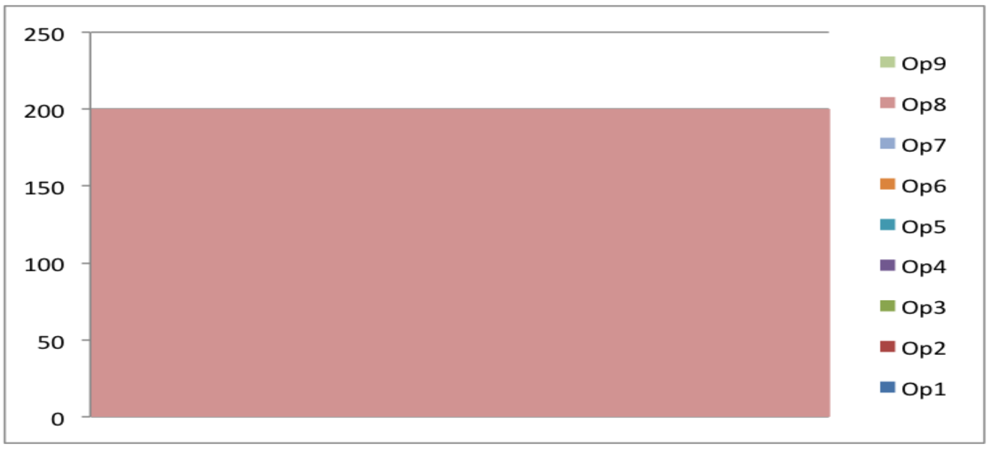
FIGURE D.2: Operator application and best fitness graphs for DE-DDQN3. Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"

f05 i01





f07 i01



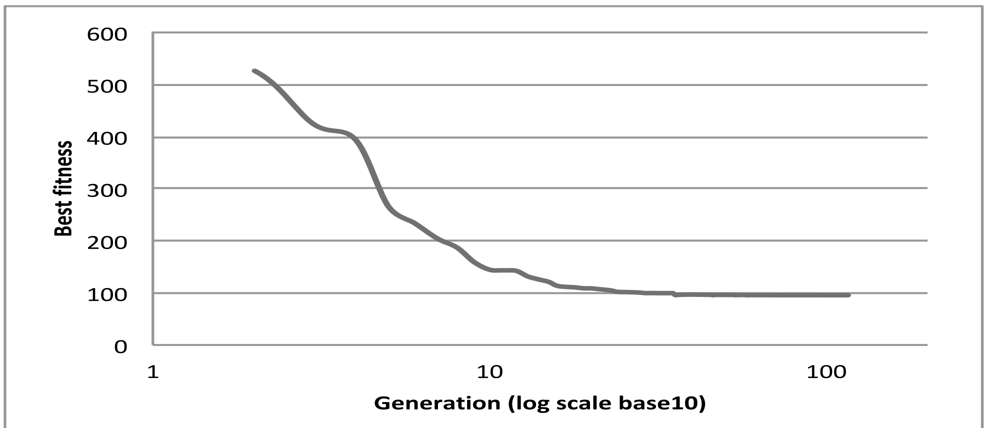
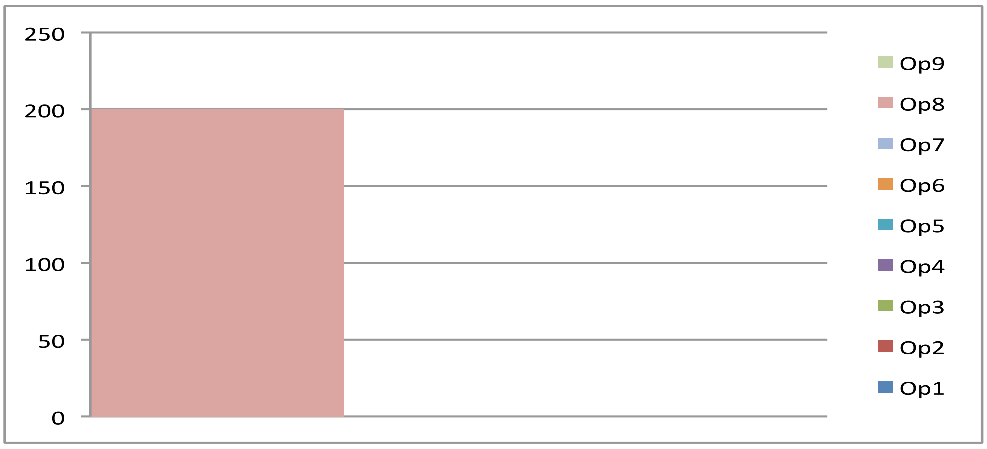
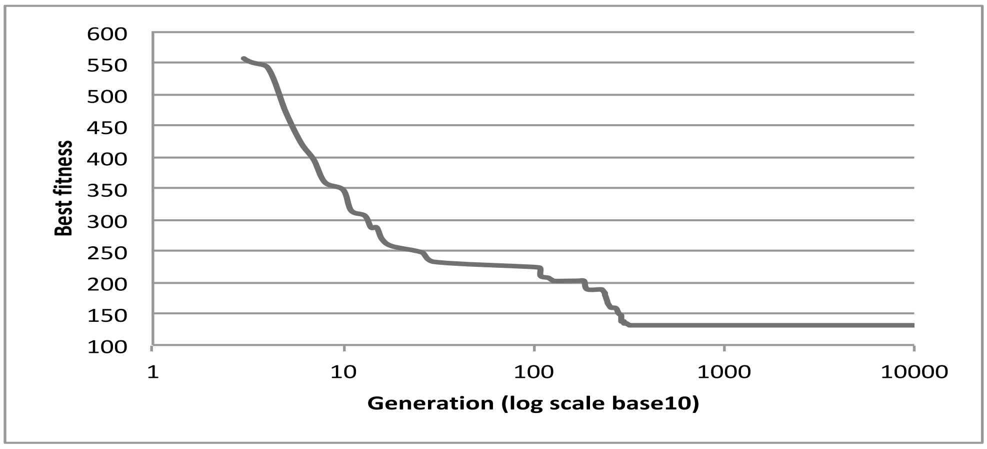


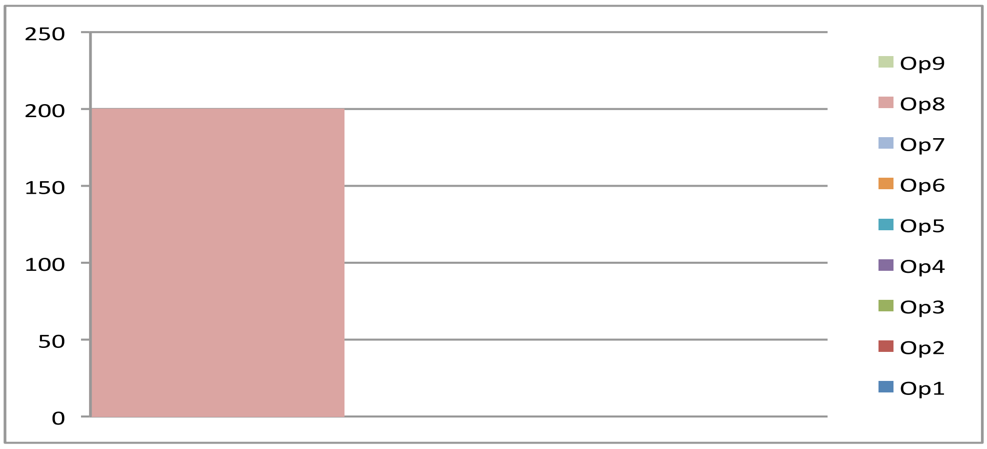
FIGURE D.2 (cont.): Operator application and best fitness graphs for DE-DDQN3. Op1: “rand/2”, Op2: “best/1”, Op3: “current-to-best/1”, Op4: “best/2”, Op5: “rand/1”, Op6: “rand-to-best/2”, Op7: “curr-to-rand/1”, op8: “curr-to-pbest/1”, Op9: “curr-to-pbest/1(archived)”

f04 i02





f08 i10



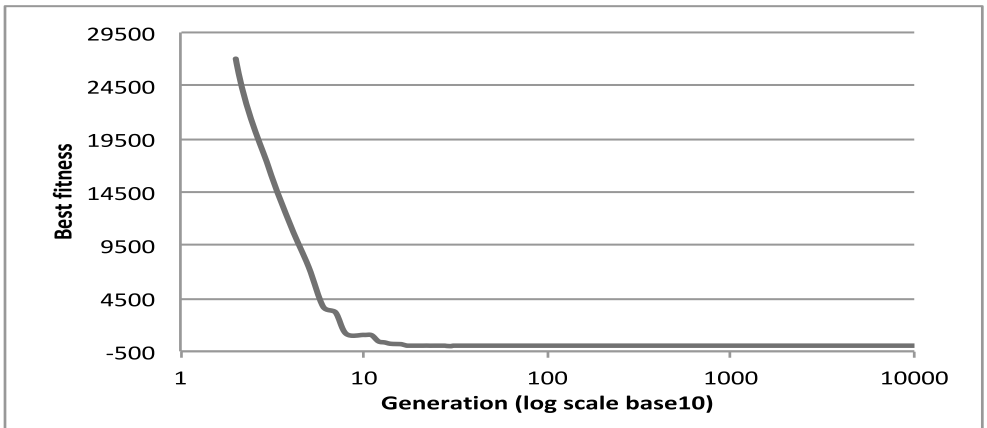
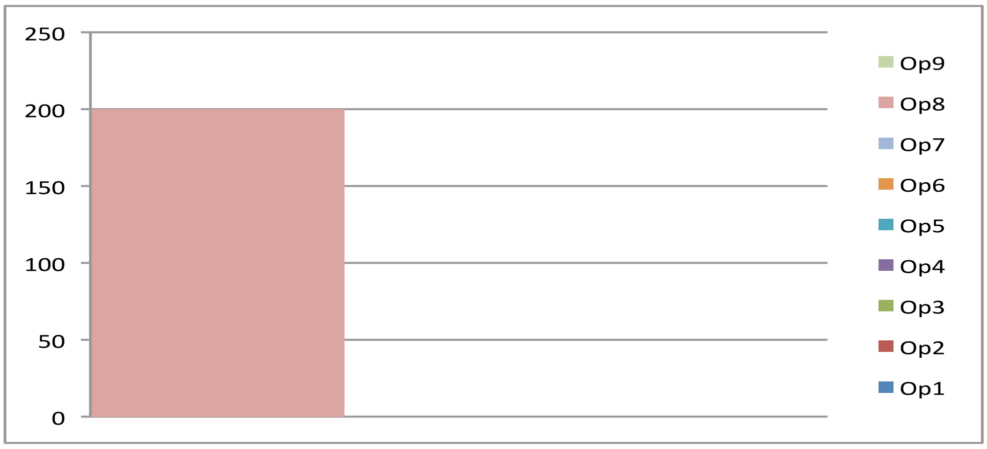
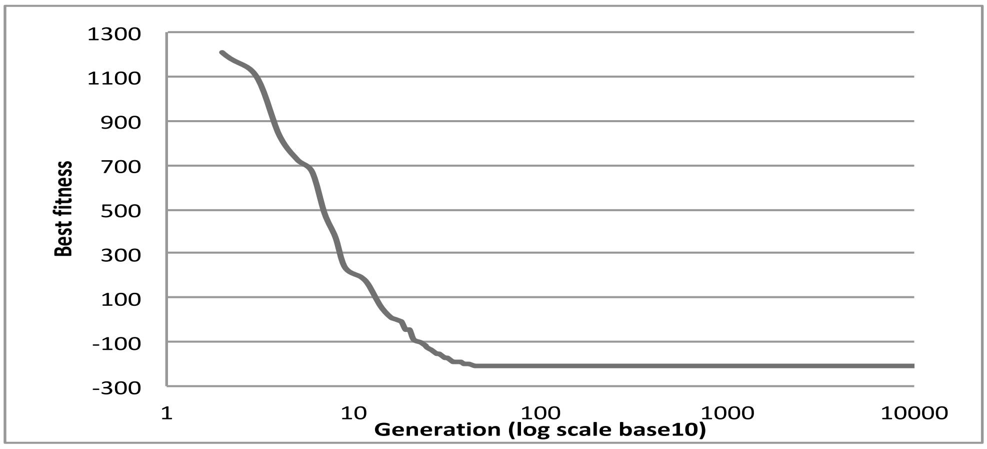


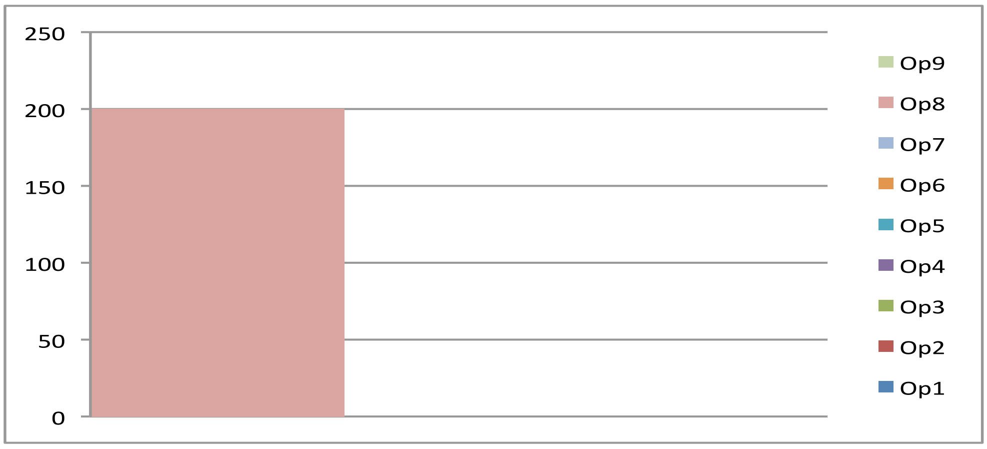
FIGURE D.2 (cont.): Operator application and best fitness graphs for DE-DDQN3. Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"

f13 i14





f17 i02



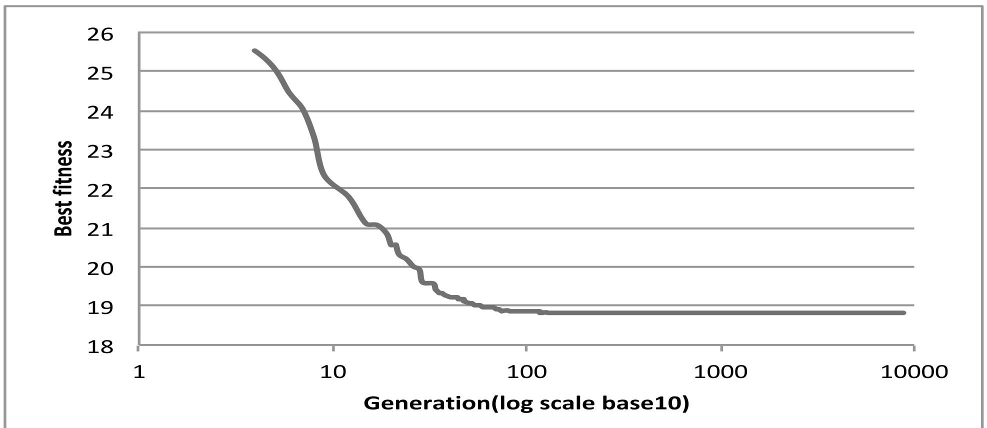
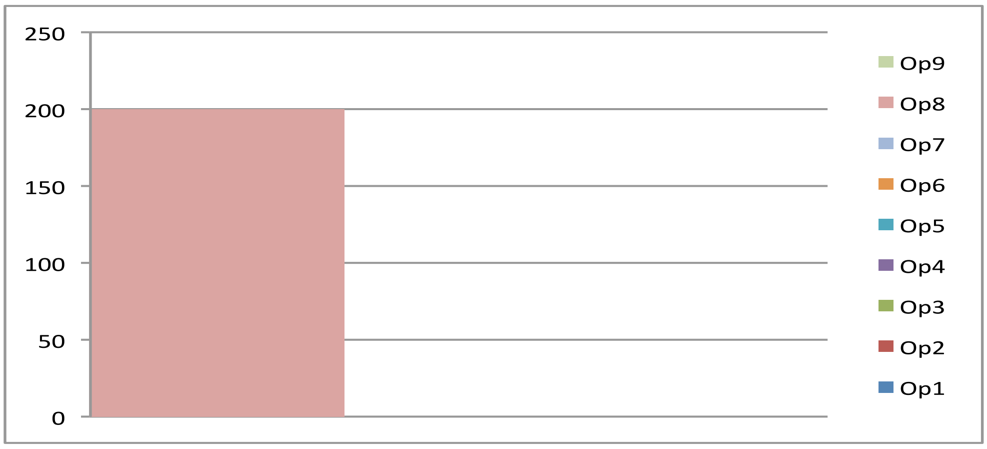


FIGURE D.2 (cont.): Operator application and best fitness graphs for DE-DDQN3. Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"

f23 i04



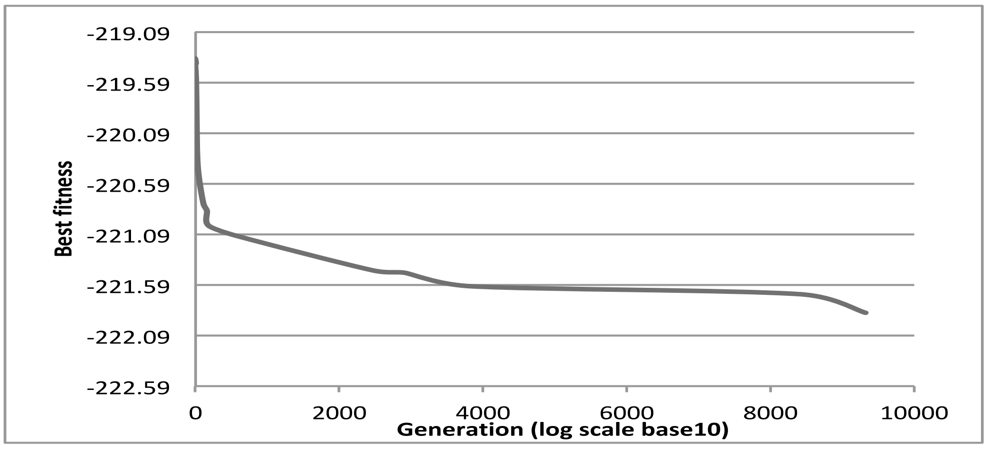
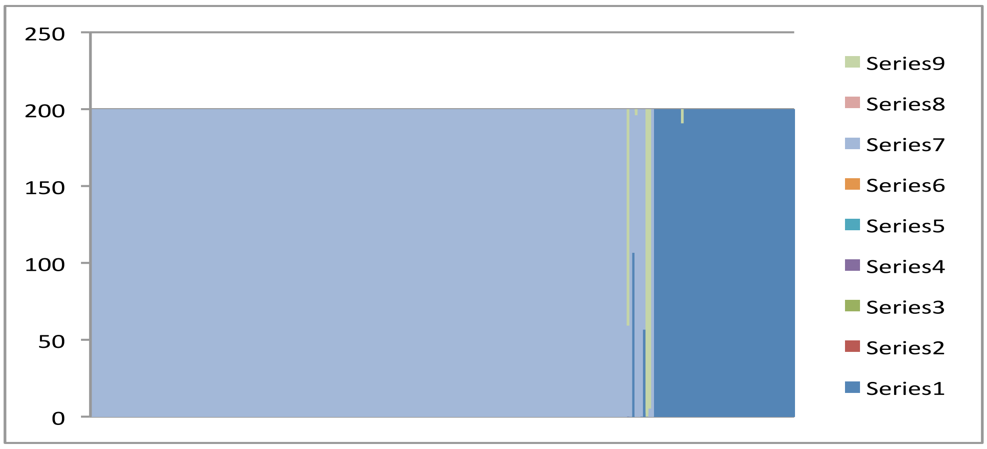
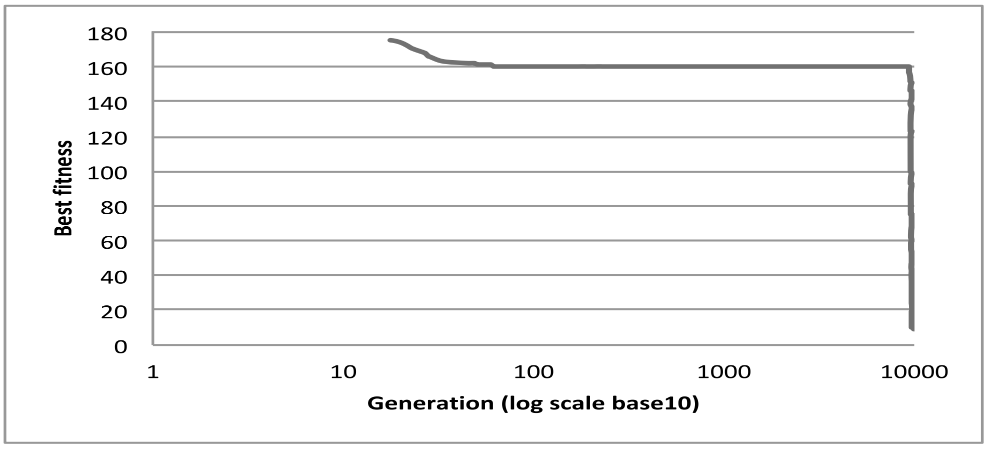


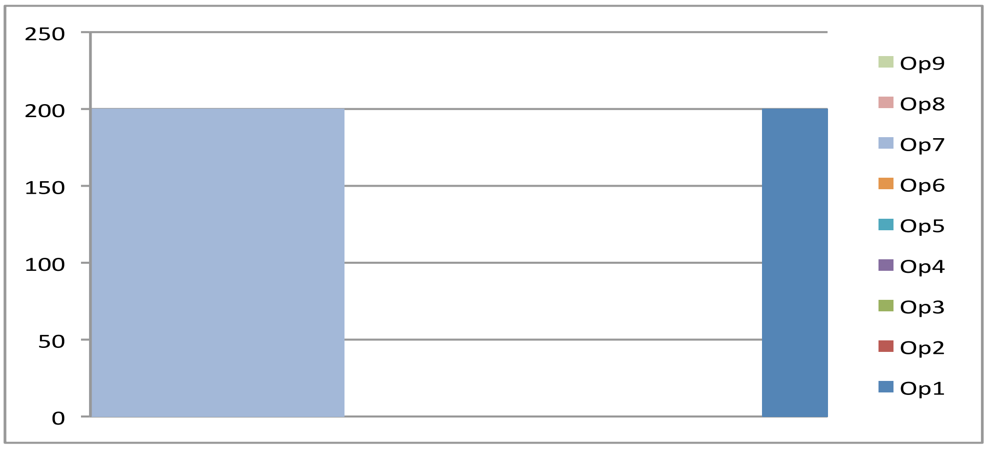
FIGURE D.3: Operator application and best fitness graphs for DE-DDQN4. Op1: “rand/2”, Op2: “best/1”, Op3: “current-to-best/1”, Op4: “best/2”, Op5: “rand/1”, Op6: “rand-to-best/2”, Op7: “curr-to-rand/1”, op8: “curr-to-pbest/1”, Op9: “curr-to-pbest/1(archived)”

f05 i01





f07 i01



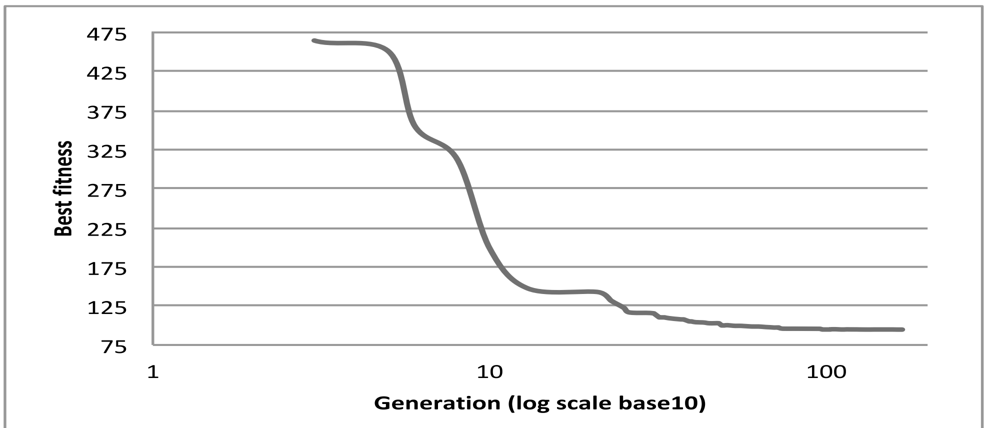
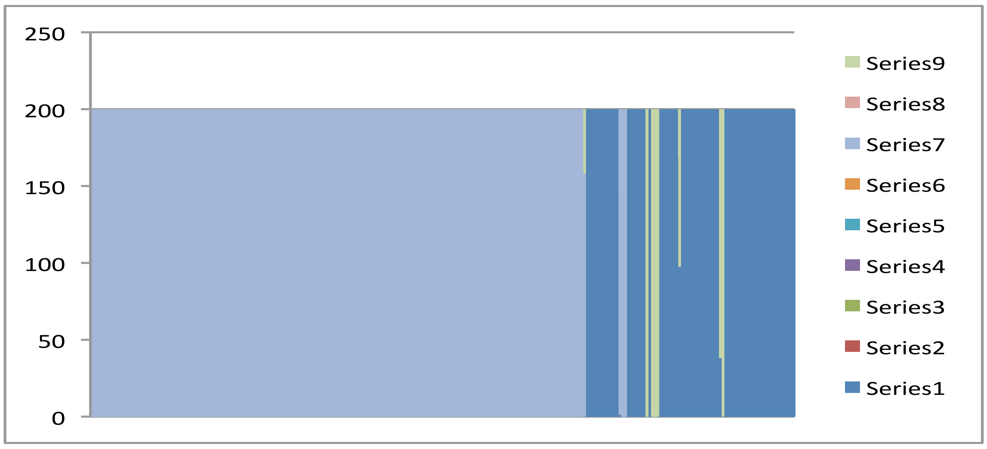
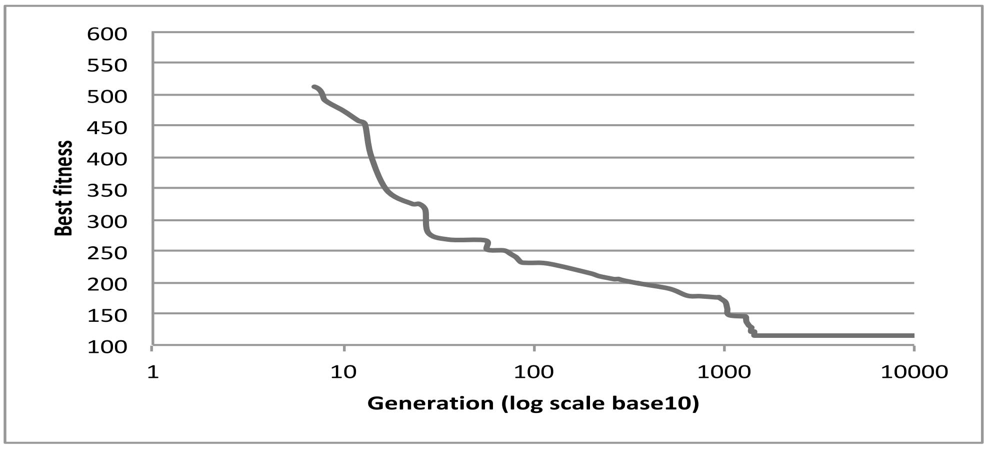


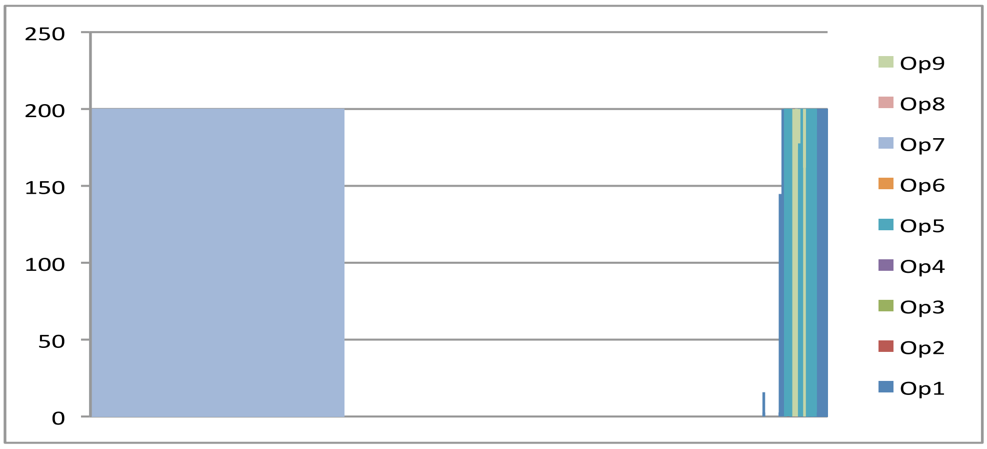
FIGURE D.3 (cont.): Operator application and best fitness graphs for DE-DDQN4. Op1: “rand/2”, Op2: “best/1”, Op3: “current-to-best/1”, Op4: “best/2”, Op5: “rand/1”, Op6: “rand-to-best/2”, Op7: “curr-to-rand/1”, op8: “curr-to-pbest/1”, Op9: “curr-to-pbest/1(archived)”

f04 i02





f08 i10



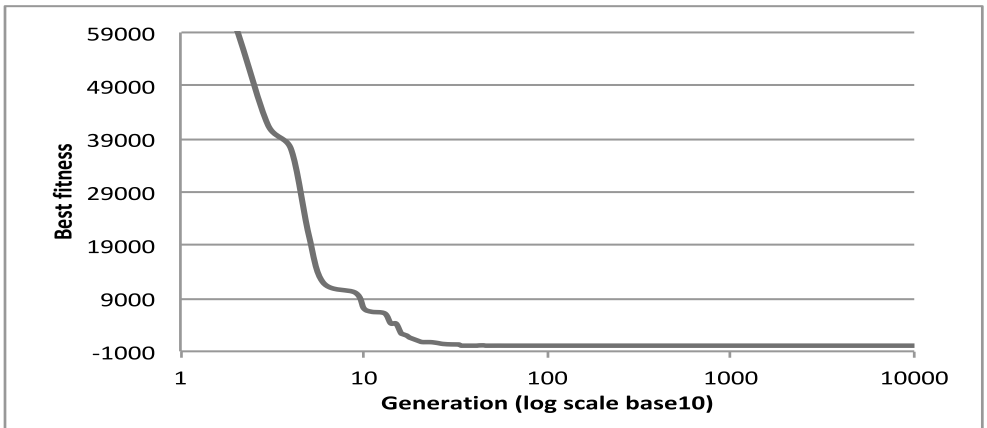
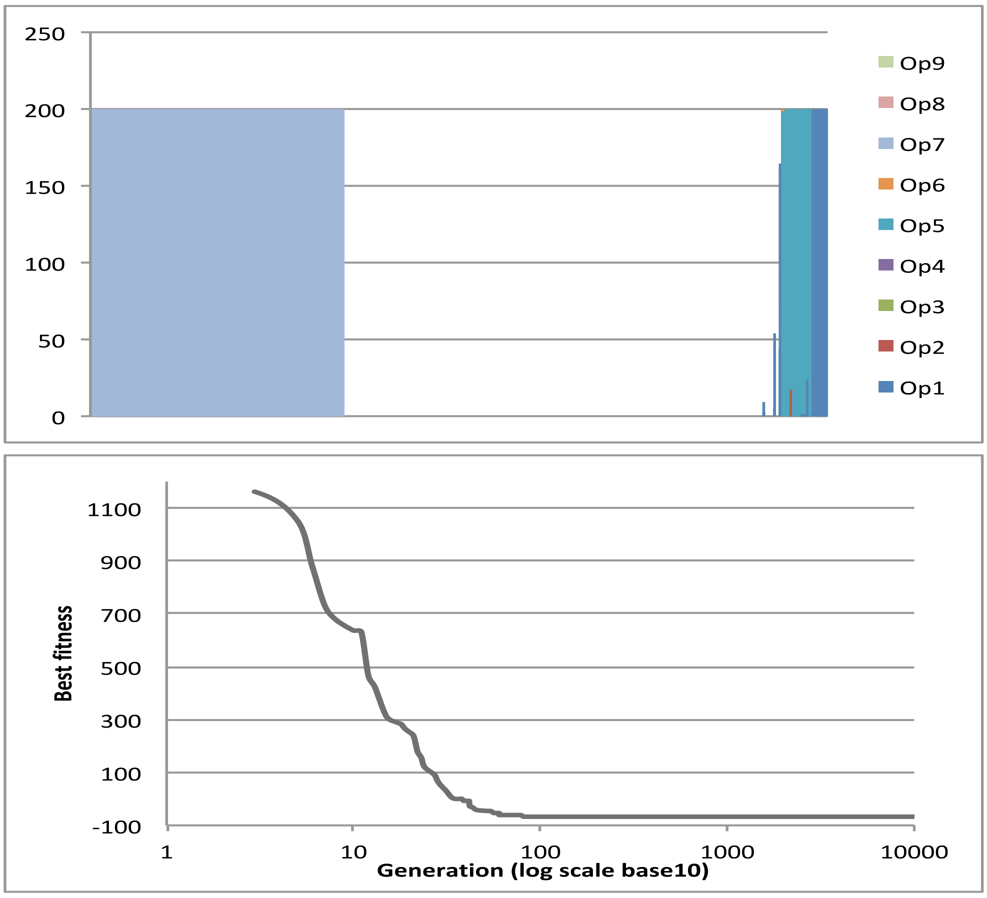


FIGURE D.3 (cont.): Operator application and best fitness graphs for DE-DDQN4. Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"

f13 i14



f17 i02

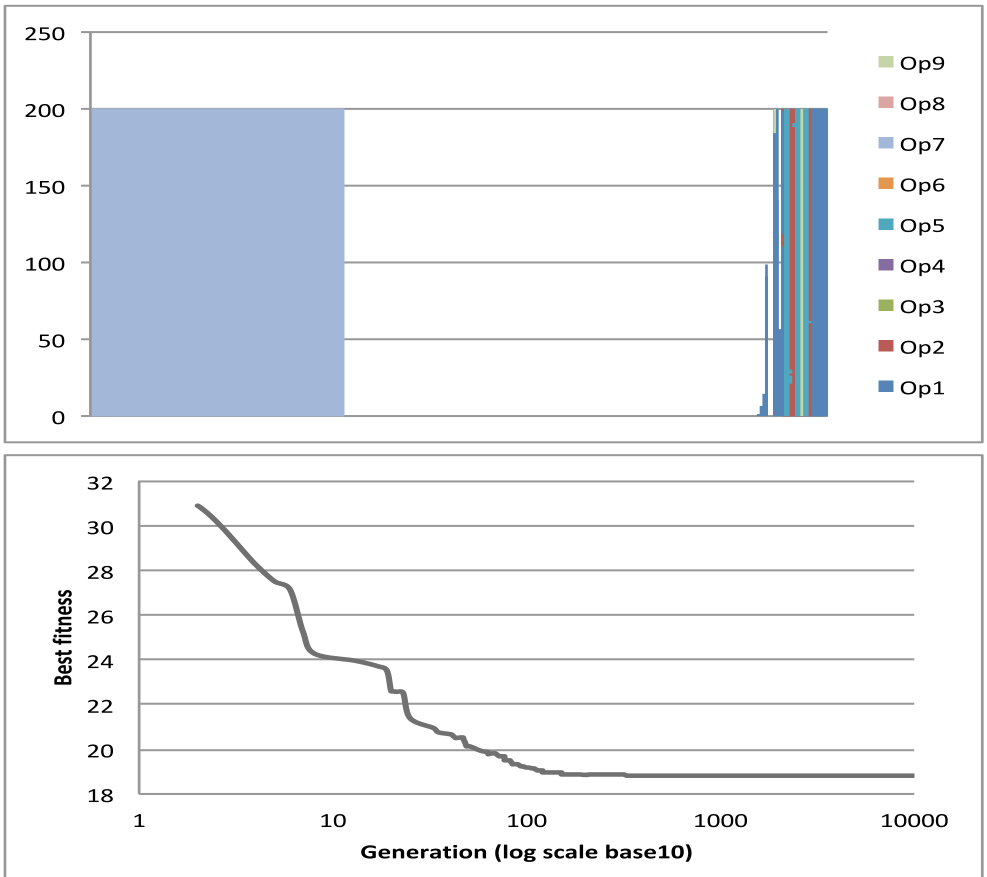
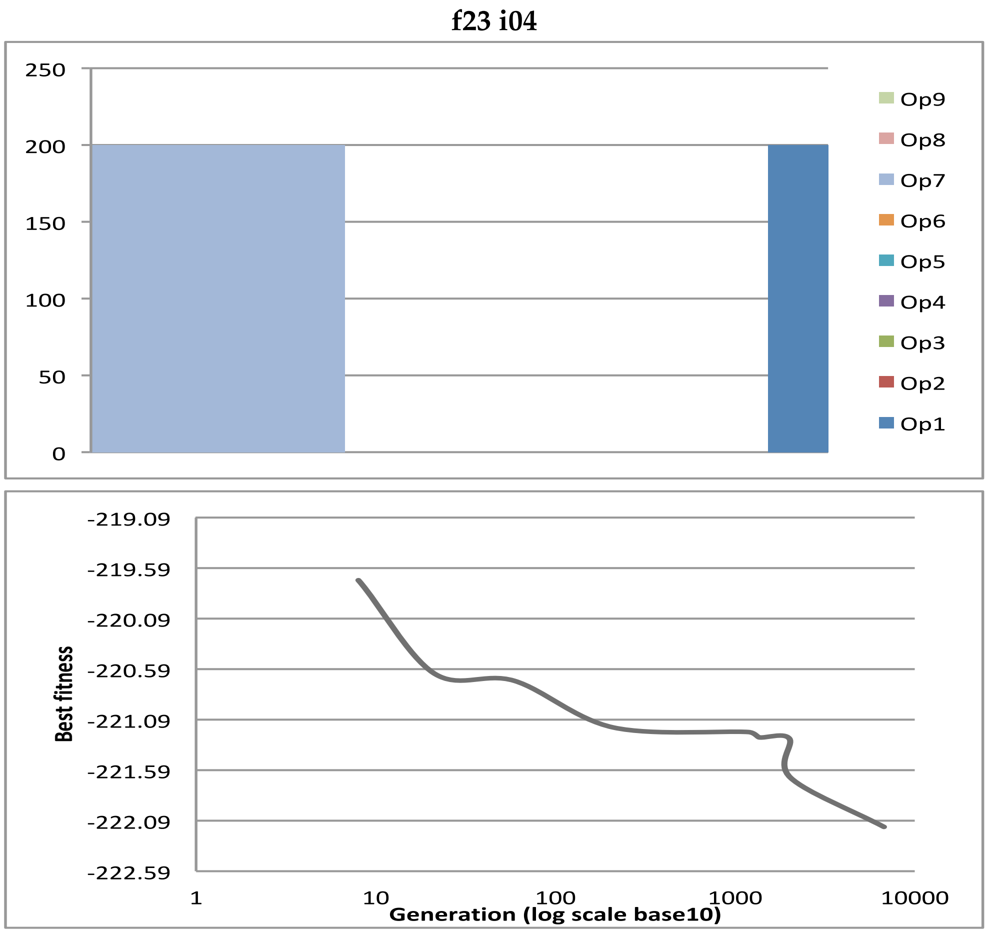


FIGURE D.3 (cont.): Operator application and best fitness graphs for DE-DDQN4. Op1: "rand/2", Op2: "best/1", Op3: "current-to-best/1", Op4: "best/2", Op5: "rand/1", Op6: "rand-to-best/2", Op7: "curr-to-rand/1", op8: "curr-to-pbest/1", Op9: "curr-to-pbest/1(archived)"



Bibliography

[1] Emile H. L. Aarts, Jan H. M. Korst, and Wil Michiels. "Simulated Annealing". In: *Search Methodologies*. Springer, 2005, pp. 187–210.

[2] Aldeida Aleti. "An adaptive approach to controlling parameters of evolutionary algorithms". In: *Swinburne University of Technology* (2012).

[3] Aldeida Aleti and Lars Grunske. "Test data generation with a Kalman filter-based adaptive genetic algorithm". In: *Journal of Systems and Software* 103 (2015), pp. 343–352.

[4] Aldeida Aleti and Irene Moser. "A systematic literature review of adaptive parameter control methods for evolutionary algorithms". In: *ACM Comput. Surv.* 49.3, Article 56 (Oct. 2016), p. 35.

[5] Aldeida Aleti and Irene Moser. "Predictive parameter control". In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM. 2011, pp. 561–568.

[6] Jaime Alvarez-Gallegos, Carlos Villar, and Edgar Flores. "Evolutionary dynamic optimization of a continuously variable transmission for mechanical efficiency maximization". In: *MICAI 2005: Advances in Artificial Intelligence* 3789 (2005), pp. 1093–1102.

[7] Peter J Angeline. "Adaptive and self-adaptive evolutionary computations". In: *Computational intelligence: a dynamic systems perspective*. Citeseer. 1995.

[8] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. "A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms". In: 2009, pp. 142–157.

[9] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-time analysis of the multiarmed bandit problem". In: 47.2-3 (2002), pp. 235–256.

[10] Anne Auger and Nikolaus Hansen. “A restart CMA evolution strategy with increasing population size”. In: *IEEE CEC*. Piscataway, NJ: IEEE Press, Sept. 2005, pp. 1769–1776.

[11] Anne Auger and Nikolaus Hansen. “Performance evaluation of an advanced local search evolutionary algorithm”. In: *IEEE CEC*. Piscataway, NJ: IEEE Press, Sept. 2005, pp. 1777–1784.

[12] Th Bäck, Agoston Eiben, and Nikolai van der Vaart. “An emperical study on GAs “without parameters””. In: *Parallel Problem Solving from Nature PPSN VI*. Springer. 2000, pp. 315–324.

[13] Thomas Back. “The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm”. In: *Proc. 2nd Conference of Parallel Problem Solving from Nature, 1992*. Elsevier Science Publishers. 1992.

[14] Prasanna Balaprakash, Mauro Birattari, and Thomas Stützle. “Improvement Strategies for the F-Race Algorithm: Sampling Design and Iterative Refinement”. In: *Hybrid Metaheuristics*. Ed. by Thomas Bartz-Beielstein, María J. Blesa, Christian Blum, Boris Naujoks, Andrea Roli, Günther Rudolph, and M. Sampels. Vol. 4771. LNCS. Springer, 2007, pp. 108–122.

[15] Helio JC Barbosa and AM Sá. “On adaptive operator probabilities in real coded genetic algorithms”. In: *XX Intl. Conf. of the Chilean Computer Science Society*. 2000.

[16] Thomas Bartz-Beielstein. *Experimental Research in Evolutionary Computation: The New Experimentalism*. Berlin, Germany: Springer, 2006.

[17] Thomas Bartz-Beielstein, C. Lasarczyk, and Mike Preuss. “Sequential Parameter Optimization”. In: *IEEE CEC*. Piscataway, NJ: IEEE Press, Sept. 2005, pp. 773–780.

[18] Bahriye Basturk. “An artificial bee colony (ABC) algorithm for numeric function optimization”. In: *IEEE Swarm Intelligence Symposium, Indianapolis, IN, USA, 2006*. 2006.

[19] Mauro Birattari, Thomas Stützle, Luís Paquete, and Klaus Varrentrapp. "A Racing Algorithm for Configuring Metaheuristics". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002*. Ed. by W. B. Langdon et al. Morgan Kaufmann Publishers, San Francisco, CA, 2002, pp. 11–18.

[20] Aymeric Blot, Manuel López-Ibáñez, Marie-Eléonore Kessaci-Marmion, and Laetitia Jourdan. "New Initialisation Techniques for Multi-Objective Local Search: Application to the Bi-objective Permutation Flowshop". In:

[21] Arina Buzdalova, Vladislav Kononov, and Maxim Buzdalov. "Selecting evolutionary operators using reinforcement learning: initial explorations". In: *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2014, pp. 1033–1036.

[22] Fei Chen, Yang Gao, Zhao-qian Chen, and Shi-fu Chen. "SCGA: Controlling genetic algorithms with sarsa (0)". In: *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*. Vol. 1. IEEE. 2005, pp. 1177–1183.

[23] Ji-Pyng Chiou and Feng-Sheng Wang. "A hybrid method of differential evolution with application to optimal control problems of a bioprocess system". In: *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*. IEEE, 1998, pp. 627–632.

[24] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. "Quantifying generalization in reinforcement learning". In: *arXiv preprint arXiv:1812.02341* (2018).

[25] Pietro Consoli and Xin Yao. "Diversity-driven selection of multiple crossover operators for the capacitated arc routing problem". In: *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer. 2014, pp. 97–108.

[26] Dave Corne, Peter Ross, and Hsiao-Lan Fang. *Ga research note 7: Fast practical evolutionary timetabling*. Tech. rep. Technical report, University of Edinburgh Department of Artificial Intelligence, 1994.

[27] S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil. "Using Experimental Design to Find Effective Parameter Settings for Heuristics". In: *J. Heuristics* 7.1 (2001), pp. 77–97.

[28] Luis DaCosta, Alvaro Fialho, Marc Schoenauer, and Michèle Sebag. "Adaptive operator selection with dynamic multi-armed bandits". In: *Proceedings of the 10th annual conference on Genetic and evolutionary computation*. ACM. 2008, pp. 913–920.

[29] Swagatam Das, Ajith Abraham, and Amit Konar. "Automatic clustering using an improved differential evolution algorithm". In: *IEEE Transactions on systems, man, and cybernetics-Part A: Systems and Humans* 38.1 (2008), pp. 218–237.

[30] Swagatam Das, Sankha Subhra Mullick, and Ponnuthurai N. Suganthan. "Recent advances in differential evolution–An updated survey". In: 27 (2016), pp. 1–30.

[31] Lawrence Davis. "Adapting operator probabilities in genetic algorithms". In: *Proc. 3rd International Conference on Genetic Algorithms, 1989*. 1989.

[32] Kenneth De Jong. "Parameter setting in EAs: a 30 year perspective". In: *Parameter setting in evolutionary algorithms*. Springer, 2007, pp. 1–18.

[33] Kalyanmoy Deb and Hans-Georg Beyer. "Self-adaptive genetic algorithms with simulated binary crossover". In: *Evolutionary computation* 9.2 (2001), pp. 197–221.

[34] Kalyanmoy Deb and Debayan Deb. "Analysing mutation schemes for real-parameter genetic algorithms". In: *Intern. J. Artif. Intell. Soft. Comput.* 4.1 (2014), pp. 1–28.

[35] Giacomo Di Tollo, Frédéric Lardeux, Jorge Maturana, and Frédéric Saubion. "An experimental study of adaptive control for evolutionary algorithms". In: *Applied Soft Computing* 35 (2015), pp. 359–372.

[36] Marco Dorigo. "Optimization, Learning and Natural Algorithms". In Italian. PhD thesis. Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.

[37] Russell Eberhart and James Kennedy. "Particle swarm optimization". In: *Proceedings of the IEEE international conference on neural networks*. Vol. 4. Citeseer. 1995, pp. 1942–1948.

[38] Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[39] AE Eiben, Mark Horvath, Wojtek Kowalczyk, and Martijn C Schut. "Reinforcement learning for online control of evolutionary algorithms". In: *International Workshop on Engineering Self-Organising Applications*. Springer. 2006, pp. 151–160.

[40] Agoston E. Eiben, Robert Hinterding, and Zbigniew Michalewicz. "Parameter Control in Evolutionary Algorithms". In: *IEEE Trans. Evol. Comput.* 3.2 (1999), pp. 124–141.

[41] Agoston E. Eiben and S. K. Smit. "Parameter Tuning for Configuring and Analyzing Evolutionary Algorithms". In: 1.1 (2011), pp. 19–31.

[42] Agoston E. Eiben, Zbigniew Michalewicz, Marc Schoenauer, and James E. Smith. "Parameter Control in Evolutionary Algorithms". In: *Parameter Setting in Evolutionary Algorithms*. Ed. by F. Lobo, C. F. Lima, and Zbigniew Michalewicz. Berlin, Germany: Springer, 2007, pp. 19–46.

[43] Agoston Endre Eiben and Selmar K Smit. "Evolutionary algorithm parameters and methods to tune them". In: *Autonomous search*. Springer, 2011, pp. 15–36.

[44] Jesse Farebrother, Marlos C Machado, and Michael Bowling. "Generalization and Regularization in DQN". In: *arXiv preprint arXiv:1810.00123* (2018).

[45] Álvaro Fialho. "Adaptive operator selection for optimization". PhD thesis. 2010.

[46] Álvaro Fialho. "Adaptive operator selection for optimization". PhD thesis. Université Paris Sud-Paris XI, 2010.

[47] Álvaro Fialho, Marc Schoenauer, and Michèle 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*. ACM. 2009, pp. 779–786.

[48] Álvaro Fialho, Marc Schoenauer, and Michèle 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: *GECCO (Companion)*. Ed. by Martin Pelikan and Jürgen Branke. New York, NY: ACM Press, 2010, pp. 1535–1542.

[49] Álvaro Fialho, Marc Schoenauer, and Michèle Sebag. "Toward comparison-based adaptive operator selection". In: *GECCO*. Ed. by Martin Pelikan and Jürgen Branke. New York, NY: ACM Press, 2010, pp. 767–774.

[50] Álvaro Fialho, Luis Da Costa, Marc Schoenauer, and Michèle Sebag. "Extreme value based adaptive operator selection". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2008, pp. 175–184.

[51] Terence C Fogarty. "Varying the probability of mutation in the genetic algorithm". In: *Proceedings of the 3rd International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers Inc. 1989, pp. 104–109.

[52] Johan Fredriksson, Kristian Sandström, and Mikael Åkerholm. "Optimizing resource usage in component-based real-time systems". In: *International Symposium on Component-based Software Engineering*. Springer. 2005, pp. 49–65.

[53] Guenther Fuellerer, Karl F. Doerner, Richard F. Hartl, and Manuel Iori. "Ant colony optimization for the two-dimensional loading vehicle routing problem". In: *Comput. Oper. Res.* 36.3 (2009), pp. 655–673.

[54] José García-Nieto, Ana Carolina Olivera, and Enrique Alba. In: ().

[55] Matthew S. Gibbs, Graeme C. Dandy, Holger R. Maier, and John B. Nixon. "Calibrating genetic algorithms for water distribution system optimisation". In: *7th Annual Symposium on Water Distribution Systems Analysis*. ASCE. May 2005.

[56] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley, 1989.

[57] David E. Goldberg. “Probability matching, the magnitude of reinforcement, and classifier system bidding”. In: 5.4 (1990), pp. 407–425.

[58] Wenyin Gong and Zhihua Cai. “Adaptive parameter selection for strategy adaptation in differential evolution”. In: *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*. ACM. 2011, pp. 111–112.

[59] Wenyin Gong, Álvaro Fialho, and Zhihua Cai. “Adaptive strategy selection in differential evolution”. In: *GECCO*. Ed. by Martin Pelikan and Jürgen Branke. New York, NY: ACM Press, 2010, pp. 409–416.

[60] Wenyin Gong, Zhihua Cai, Charles X Ling, and Hui Li. “Enhanced differential evolution with adaptive strategies for numerical optimization”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 41.2 (2010), pp. 397–413.

[61] J. J. Grefenstette. “Optimization of Control Parameters for Genetic Algorithms”. In: 16.1 (1986), pp. 122–128.

[62] Nikolaus Hansen. “Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed”. In: *GECCO (Companion)*. Ed. by Franz Rothlauf. New York, NY: ACM Press, 2009, pp. 2389–2396.

[63] Nikolaus Hansen and Stefan Kern. “Evaluating the CMA evolution strategy on multimodal test functions”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2004, pp. 282–291.

[64] Nikolaus Hansen and A. Ostermeier. “Completely derandomized self-adaptation in evolution strategies”. In: *Evol. Comput.* 9.2 (2001), pp. 159–195.

[65] Nikolaus Hansen and Raymond Ros. “Benchmarking a weighted negative covariance matrix update on the BBOB-2010 noiseless testbed”. In: *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*. ACM. 2010, pp. 1673–1680.

[66] Nikolaus Hansen, A. Auger, Olaf Mersmann, Tea Tusar, and Dimo Brockhoff. “COCO: A platform for comparing continuous optimizers in a black-box setting”. In: *1603.08785* (2016).

[67] Nikolaus Hansen, S. Finck, R. Ros, and A. Auger. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions*. Tech. rep. RR-6829. Updated February 2010. INRIA, France, 2009.

[68] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning". In:

[69] Francisco Herrera and Manuel Lozano. "Adaptive genetic operators based on coevolution with fuzzy behaviors". In: *IEEE Transactions on Evolutionary computation* 5.2 (2001), pp. 149–165.

[70] Jürgen Hesser and Reinhard Männer. "Towards an optimal mutation probability for genetic algorithms". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 1990, pp. 23–32.

[71] Robert Hinterding, Zbigniew Michalewicz, and T Peachey. "Self-adaptive genetic algorithm for numeric functions". In: *Parallel Problem Solving from Nature—PPSN IV* (1996), pp. 420–429.

[72] Tzung-Pei Hong, Hong-Shung Wang, and Wei-Chou Chen. "Simultaneously applying multiple mutation operators in genetic algorithms". In: *Journal of heuristics* 6.4 (2000), pp. 439–455.

[73] R Howard. *Dynamic Probabilistic Systems: Semi-Markov and Decision Systems, volume II*. 1971.

[74] Z Hussain, P Auer, N Cesa-Bianchi, L Newnham, and J Shawe-Taylor. *Exploration vs. exploitation pascal challenge*. 2006.

[75] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. "Sequential Model-Based Optimization for General Algorithm Configuration". In: *Learning and Intelligent Optimization, 5th International Conference, LION 5*. Ed. by Carlos A. Coello Coello. Vol. 6683. LNCS. Springer, 2011, pp. 507–523.

[76] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. "ParamILS: An Automatic Algorithm Configuration Framework". In: *J. Artif. Intell. Res.* 36 (Oct. 2009), pp. 267–306.

[77] Christian Igel and Martin Kreutz. “Operator adaptation in evolutionary computation and its application to structure optimization of neural networks”. In: *Neurocomputing* 55.1-2 (2003), pp. 347–361.

[78] Christian Igel and Martin Kreutz. “Using fitness distributions to improve the evolution of learning structures”. In: *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*. Vol. 3. IEEE. 1999, pp. 1902–1909.

[79] Srikanth K. Iyer and Barkha Saxena. “Improved genetic algorithm for the permutation flowshop scheduling problem”. In: *Comput. Oper. Res.* 31.4 (2004), pp. 593–606.

[80] Cezary Z Janikow and Zbigniew Michalewicz. “An experimental comparison of binary and floating point representations in genetic algorithms.” In: *ICGA*. 1991, pp. 31–36.

[81] M Johns, H Mahmoud, D Walker, N Ross, Edward C Keedwell, and D Savic. “Augmented Evolutionary Intelligence: Combining Human and Evolutionary Design for Water Distribution Network Optimisation”. In: (2019).

[82] Alan W. Johnson and Sheldon H. Jacobson. “On the Convergence of Generalized Hill Climbing Algorithms”. In: 119.1 (2002), pp. 37–57.

[83] P. W. Jowitt and G. Germanopoulos. “Optimal pump scheduling in water supply networks”. In: 118.4 (1992), pp. 406–422.

[84] Bryant A Julstrom. “Adaptive operator probabilities in a genetic algorithm that applies three operators”. In: *Proceedings of the 1997 ACM symposium on Applied computing*. ACM. 1997, pp. 233–238.

[85] Bryant A Julstrom. “An inquiry into the behavior of adaptive operator probabilities in steady-state genetic algorithms”. In: *Proceedings of the Second Nordic Workshop on Genetic Algorithms and their Applications*. August, 1996, pp. 15–26.

[86] Bryant A. Julstrom. “What Have You Done for Me Lately? Adapting Operator Probabilities in a Steady-State Genetic Algorithm”. In: *ICGA*. Ed. by Larry J. Eshelman. Morgan Kaufmann Publishers, San Francisco, CA, 1995, pp. 81–87.

[87] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. "Illuminating generalization in deep reinforcement learning through procedural level generation". In: *arXiv preprint arXiv:1806.10729* (2018).

[88] Dervis Karaboga. *An idea based on honey bee swarm for numerical optimization*. Tech. rep. Technical report-tr06, Erciyes university, engineering faculty, computer . . ., 2005.

[89] Dervis Karaboga and Bahriye Akay. "A comparative study of artificial bee colony algorithm". In: *Applied mathematics and computation* 214.1 (2009), pp. 108–132.

[90] Dervis Karaboga and Bahriye Akay. "Artificial bee colony (ABC) algorithm on training artificial neural networks". In: *2007 IEEE 15th Signal Processing and Communications Applications*. IEEE. 2007, pp. 1–4.

[91] Dervis Karaboga, Bahriye Akay, and Celal Ozturk. "Artificial bee colony (ABC) optimization algorithm for training feed-forward neural networks". In: *International conference on modeling decisions for artificial intelligence*. Springer. 2007, pp. 318–329.

[92] Dervis Karaboga and Bahriye Basturk. "Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems". In: *International fuzzy systems association world congress*. Springer. 2007, pp. 789–798.

[93] Dervis Karaboga and Bahriye Basturk. "On the performance of artificial bee colony (ABC) algorithm". In: *Applied soft computing* 8.1 (2008), pp. 687–697.

[94] Dervis Karaboga and Beyza Gorkemli. "A quick artificial bee colony (qABC) algorithm and its performance on optimization problems". In: *Applied Soft Computing* 23 (2014), pp. 227–238.

[95] Dervis Karaboga and Ebubekir Kaya. "An adaptive and hybrid artificial bee colony algorithm (aABC) for ANFIS training". In: *Applied Soft Computing* 49 (2016), pp. 423–436.

[96] Dervis Karaboga, Beyza Gorkemli, Celal Ozturk, and Nurhan Karaboga. "A comprehensive survey: artificial bee colony (ABC) algorithm and applications". In: *Artificial Intelligence Review* 42.1 (2014), pp. 21–57.

[97] Nurhan Karaboga. "A new design method based on artificial bee colony algorithm for digital IIR filters". In: *Journal of the Franklin Institute* 346.4 (2009), pp. 328–348.

[98] NURHAN KARABOĞA and Mehmet Bahadir Cetinkaya. "A novel and efficient algorithm for adaptive filtering: artificial bee colony algorithm". In: *Turkish Journal of Electrical Engineering & Computer Sciences* 19.1 (2011), pp. 175–190.

[99] Giorgos Karafotias, Agoston Endre Eiben, and Mark Hoogendoorn. "Generic parameter control with reinforcement learning". In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2014, pp. 1319–1326.

[100] Giorgos Karafotias, Mark Hoogendoorn, and AE Eiben. "Evaluating reward definitions for parameter control". In: *European Conference on the Applications of Evolutionary Computation*. Springer. 2015, pp. 667–680.

[101] Giorgos Karafotias, Mark Hoogendoorn, and Agoston E. Eiben. "Parameter Control in Evolutionary Algorithms: Trends and Challenges". In: *IEEE Trans. Evol. Comput.* 19.2 (2015), pp. 167–187.

[102] Giorgos Karafotias, Mark Hoogendoorn, and Berend Weel. "Comparing generic parameter controllers for EAs". In: *Foundations of Computational Intelligence (FOCI), 2014 IEEE Symposium on*. IEEE. 2014, pp. 46–53.

[103] Giorgos Karafotias, Selmar K Smit, and AE Eiben. "A generic approach to parameter control". In: *European Conference on the Applications of Evolutionary Computation*. Springer. 2012, pp. 366–375.

[104] Eric Kee, Sarah Airey, and Walling Cyre. "An adaptive genetic algorithm". In: pp. 391–397.

[105] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *Arxiv preprint arXiv:1412.6980 [cs.LG]* (2014).

[106] Joshua D. Knowles. "Local-Search and Hybrid Evolutionary Algorithms for Pareto Optimization". PhD thesis. University of Reading, UK, 2002.

[107] Oliver Kramer. "Evolutionary self-adaptation: a survey of operators and strategy parameters". In: *Evolutionary Intelligence* 3.2 (2010), pp. 51–65.

[108] Jouni Lampinen. "A constraint handling approach for the differential evolution algorithm". In: *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*. Vol. 2. IEEE, 2002, pp. 1468–1473.

[109] Jianjun David Li. "A two-step rejection procedure for testing multiple hypotheses". In: *Journal of Statistical Planning and Inference* 138.6 (2008), pp. 1521–1527.

[110] Tianjun Liao and Thomas Stützle. "Benchmark results for a simple hybrid algorithm on the CEC 2013 benchmark set for real-parameter optimization". In: *Proceedings of the 2013 Congress on Evolutionary Computation (CEC 2013)*. Piscataway, NJ: IEEE Press, 2013, pp. 1938–1944.

[111] Junhong Liu and Jouni Lampinen. "A fuzzy adaptive differential evolution algorithm". In: *Soft Computing* 9.6 (2005), pp. 448–462.

[112] F. Lobo, C. F. Lima, and Zbigniew Michalewicz, eds. *Parameter Setting in Evolutionary Algorithms*. Berlin, Germany: Springer, 2007.

[113] Fernando G Lobo and David E Goldberg. "Decision making in a hybrid genetic algorithm". In: *Evolutionary Computation, 1997., IEEE International Conference on*. IEEE. 1997, pp. 121–125.

[114] Fernando G Lobo and Claudio F Lima. "Adaptive population sizing schemes in genetic algorithms". In: *Parameter setting in evolutionary algorithms*. Springer, 2007, pp. 185–204.

[115] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. "The irace package: Iterated Racing for Automatic Algorithm Configuration". In: (2016).

[116] O. Maron and Andrew Moore. "Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation". In: *Advances in Neural Information ProcessingSystems 6*. Morgan Kaufmann, 1993.

[117] Jorge Maturana, Frédéric Lardeux, and Frédéric Saubion. “Autonomous operator management for evolutionary algorithms”. In: *Journal of Heuristics* 16.6 (2010), pp. 881–909.

[118] Jorge Maturana and Frédéric Saubion. “A compass to guide genetic algorithms”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2008, pp. 256–265.

[119] Jorge Maturana, Alvaro Fialho, Frédéric Saubion, Marc Schoenauer, Frédéric Lardeux, and Michele Sebag. “Adaptive operator selection and management in evolutionary algorithms”. In: *Autonomous Search*. Springer, 2011, pp. 161–189.

[120] Jorge Maturana, Álvaro Fialho, Frédéric Saubion, Marc Schoenauer, and Michèle Sebag. “Extreme compass and dynamic multi-armed bandits for adaptive operator selection”. In: *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE. 2009, pp. 365–372.

[121] Brian Mc Ginley, John Maher, Colm O'Riordan, and Fearghal Morgan. “Maintaining healthy population diversity using adaptive crossover, mutation, and selection”. In: *IEEE Transactions on Evolutionary Computation* 15.5 (2011), pp. 692–714.

[122] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), p. 529.

[123] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. “The application of Bayesian methods for seeking the extremum”. In: *Towards global optimization* 2.117-129 (1978), p. 2.

[124] Sibylle D Muller, Nicol N Schraudolph, and Petros D Koumoutsakos. “Step size adaptation in evolution strategies using reinforcement learning”. In: *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*. Vol. 1. IEEE. 2002, pp. 151–156.

[125] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: pp. 807–814.

[126] Jens Niehaus and Wolfgang Banzhaf. "Adaption of operator probabilities in genetic programming". In: *European Conference on Genetic Programming*. Springer. 2001, pp. 325–336.

[127] Yew Soon Ong and Andy J Keane. "Meta-Lamarckian learning in memetic algorithms". In: *IEEE transactions on evolutionary computation* 8.2 (2004), pp. 99–110.

[128] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. "Assessing generalization in deep reinforcement learning". In: *arXiv preprint arXiv:1810.12282* (2018).

[129] Rebecca Parsons and Mark Johnson. "A Case Study in Experimental Design Applied to Genetic Algorithms with Applications to DNA Sequence Assembly". In: *American Journal of Mathematical and Management Sciences* 17.3-4 (1997), pp. 369–396.

[130] Martin Pelikan and Jürgen Branke, eds. *Genetic and Evolutionary Computation Conference, GECCO 2010, Proceedings, Portland, Oregon, USA, July 7-11, 2010*. New York, NY: ACM Press, 2010.

[131] James E Pettinger and Richard M Everson. "Controlling genetic algorithms with reinforcement learning". In: *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc. 2002, pp. 692–692.

[132] Kenneth Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Springer, New York, NY, 2005.

[133] *Proceedings of the 2005 Congress on Evolutionary Computation (CEC 2005)*. Piscataway, NJ: IEEE Press, Sept. 2005.

[134] Wuwen Qian, Junrui Chai, Zengguang Xu, and Ziying Zhang. "Differential evolution algorithm with multiple mutation strategies based on roulette wheel selection". In: *Applied Intelligence* (2018), pp. 1–18.

[135] A Kai Qin, Vicky Ling Huang, and Ponnuthurai N Suganthan. "Differential evolution algorithm with strategy adaptation for global numerical optimization". In: *IEEE transactions on Evolutionary Computation* 13.2 (2009), pp. 398–417.

[136] A Kai Qin and Ponnuthurai N Suganthan. "Self-adaptive differential evolution algorithm for numerical optimization". In: *Evolutionary Computation, 2005. The 2005 IEEE Congress on*. Vol. 2. IEEE. 2005, pp. 1785–1791.

[137] I. Rechenberg. "Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution". PhD thesis. Department of Process Engineering, Technical University of Berlin, 1971.

[138] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, Germany, 1973.

[139] Ingo Rechenberg. "Cybernetic solution path of an experimental problem". In: *Royal Aircraft Establishment Library Translation 1122* (1965).

[140] Colin R. Reeves. "Genetic algorithms". In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. 2nd ed. Vol. 146. International Series in Operations Research & Management Science. New York, NY: Springer, 2010. Chap. 5, pp. 109–140.

[141] Arkady Rost, Irina Petrova, and Arina Buzdalova. "Adaptive Parameter Selection in Evolutionary Algorithms by Reinforcement Learning with Dynamic Discretization of Parameter Range". In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. ACM. 2016, pp. 141–142.

[142] John Rust. "Structural estimation of Markov decision processes". In: *Handbook of Econometrics*. Vol. 4. Elsevier, 1994, pp. 3081–3143.

[143] Yoshitaka Sakurai, Kouhei Takada, Takashi Kawabe, and Setsuo Tsuruta. "A method to control parameters of evolutionary algorithms by using reinforcement learning". In: *2010 Sixth International Conference on Signal-Image Technology and Internet Based Systems*. IEEE. 2010, pp. 74–79.

[144] Tom Schaul. "Comparing natural evolution strategies to bipop-cma-es on noiseless and noisy black-box optimization testbeds". In: *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*. ACM. 2012, pp. 237–244.

[145] Hans-Paul Schwefel. "Evolutionsstrategien für die numerische Optimierung". In: *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Springer, 1977, pp. 123–176.

[146] Hans-Paul Schwefel. *Numerical optimization of computer models*. John Wiley & Sons, Inc., 1981.

[147] Mudita Sharma and Satish Chandra. "Application of Artificial Bee Colony Algorithm for numerical optimization technique". In: *2015 IEEE International Advance Computing Conference (IACC)*. IEEE. 2015, pp. 1267–1272.

[148] Mudita Sharma and Dimitar Kazakov. "Hybridisation of artificial bee colony algorithm on four classes of real-valued optimisation functions". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM. 2017, pp. 1439–1442.

[149] Mudita Sharma, Manuel López-Ibáñez, and Dimitar Kazakov. *Deep Reinforcement Learning Based Parameter Control in Differential Evolution: Supplementary material*. https://github.com/mudita11/DE-DDQN. 2019. DOI: 10.5281/zenodo.2628229.

[150] Mudita Sharma, Manuel López-Ibáñez, and Dimitar Kazakov. "Performance Assessment of Recursive Probability Matching for Adaptive Operator Selection in Differential Evolution". In: *Parallel Problem Solving from Nature - PPSN XV*. Ed. by Anne Auger, Carlos M. Fonseca, N. Lourenço, P. Machado, Luís Paquete, and Darrell Whitley. Vol. 11102. LNCS. Springer, Cham, 2018, pp. 321–333.

[151] Mudita Sharma, Manuel López-Ibáñez, and Dimitar Kazakov. *Performance Assessment of Recursive Probability Matching for Adaptive Operator Selection in Differential Evolution: Supplementary material*. https://github.com/mudita11/AOS-comparisons. 2018. DOI: 10.5281/zenodo.1257672.

[152] Alok Singh. “An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem”. In: *Applied Soft Computing* 9.2 (2009), pp. 625–631.

[153] Jim Smith and Terence C Fogarty. “Self adaptation of mutation rates in a steady state genetic algorithm”. In: *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*. IEEE. 1996, pp. 318–323.

[154] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. “Practical bayesian optimization of machine learning algorithms”. In: *Advances in neural information processing systems*. 2012, pp. 2951–2959.

[155] Jaroslaw T Stanczak, Jan J Mulawka, and Brijesh K Verma. “Genetic algorithms with adaptive probabilities of operators selection”. In: *iccima*. IEEE. 1999, p. 464.

[156] Rainer Storn and Kenneth Price. “Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces”. In: *J. Glob. Optim.* 11.4 (1997), pp. 341–359.

[157] Ponnuthurai N. Suganthan, Nikolaus Hansen, J. J. Liang, Kalyanmoy Deb, Y. P. Chen, A. Auger, and S. Tiwari. *Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization*. Tech. rep. Nanyang Technological University, Singapore, 2005.

[158] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[159] Ryoji Tanabe and Alex Fukunaga. “Success-history based parameter adaptation for differential evolution”. In: *2013 IEEE congress on evolutionary computation*. IEEE. 2013, pp. 71–78.

[160] Ryoji Tanabe and Alex Fukunaga. “Tuning differential evolution for cheap, medium, and expensive computational budgets”. In: *2015 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2015, pp. 2018–2025.

[161] Cristina Teixeira, José Covas, Thomas Stützle, and António Gaspar-Cunha. “Application of Pareto Local Search and Multi-Objective Ant Colony Algorithms to the Optimization of Co-Rotating Twin Screw Extruders”. In: *Proceedings of the EU/MEeting 2009: Debating the future: new areas of application and innovative approaches*. Ed. by Ana Viana et al. 2009, pp. 115–120.

[162] Dirk Thierens. “Adaptive strategies for operator allocation”. In: *Parameter Setting in Evolutionary Algorithms*. Ed. by F. Lobo, C. F. Lima, and Zbigniew Michalewicz. Berlin, Germany: Springer, 2007, pp. 77–90.

[163] Dirk Thierens. “An Adaptive Pursuit Strategy for Allocating Operator Probabilities”. In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2005*. Ed. by Hans-Georg Beyer and Una-May O’Reilly. New York, NY: ACM Press, 2005, pp. 1539–1546.

[164] Andrew Tuson and Peter Ross. “Adapting operator settings in genetic algorithms”. In: *Evolutionary computation* 6.2 (1998), pp. 161–184.

[165] Fatemeh Vafaee, Peter C Nelson, Chi Zhou, and Weimin Xiao. “Dynamic adaptation of genetic operators’ probabilities”. In: *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*. Springer, 2008, pp. 159–168.

[166] James M Whitacre, Tuan Q Pham, and Ruhul A Sarker. “Credit assignment in adaptive evolutionary algorithms”. In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM. 2006, pp. 1353–1360.

[167] Y-Y Wong, K-H Lee, K-S Leung, and C-W Ho. “A novel approach in parameter adaptation and diversity maintenance for genetic algorithms”. In: *Soft Computing-A Fusion of Foundations, Methodologies and Applications* 7.8 (2003), pp. 506–515.

[168] Jinghua Zhang and Ze Dong. “Parameter Combination Framework for the Differential Evolution Algorithm”. In: *Algorithms* 12.4 (2019), p. 71.

[169] Jingqiao Zhang and Arthur C. Sanderson. “JADE: adaptive differential evolution with optional external archive”. In: *IEEE Trans. Evol. Comput.* 13.5 (2009), pp. 945–958.

[170] Guopu Zhu and Sam Kwong. “Gbest-guided artificial bee colony algorithm for numerical function optimization”. In: *Applied mathematics and computation* 217.7 (2010), pp. 3166–3173.

[171] Eckart Zitzler. “Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications”. PhD thesis. ETH Zürich, Switzerland, 1999.