Universiteit Leiden

# Master Computer Science

Detecting the switch time for informed one-shot dynamic algorithm selection using the area under the ECDF curve


| | |
|---|---|
| Name: | Yiqin Lei |
| Student ID: | s2555069 |
| Date: | 28/11/2022 |
| Specialisation: | Computer Science: Data Science |
| 1st supervisor: | Furong Ye |
| 2nd supervisor: | Yingjie Fan |


Master's Thesis in Computer Science


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

# Abstract

Evolutionary Algorithms with different parameter configurations have different performance on a given problem. In the applications of evolutionary algorithms, the most important steps are configuring parameters and operators. A proper configuration of an algorithm leads to good performance on the problem at hand. In order to figure out which configuration may be the best suited of different stages of the optimization process, we follow the previous investigation about dynamic algorithm selection.

We use one-shot setting for algorithm before the run, of which we obtained from experimental data. We use area under the empirical distribution function curve as an alternative performance measure which differs from previous work that only used expected running time. We apply a policy that switches from an algorithm to another algorithm once on an optimization problem under the construction of a family of genetic algorithms. In this work, we perform empirical analyse on how the area under the curve can be used to infer the policy for dynamic algorithm selection schemes on some specific optimization problems.

**KEYWORDS:**
Genetic algorithm, Dynamic algorithm selection, Area under the curve, Black-Box Optimization, Evolutionary Computation

# Contents

# Chapter 1

# Introduction

## 1.1 Scope of the thesis

In the past decades, Evolutionary Algorithm (EA) has been highly developed with different techs that differ in genetic representation and other implementation details and the nature of the particular applied problem. This thesis focuses on Genetic Algorithm (GA) which is the most basic type of EAs that is commonly used for optimization problems.

GA seeks the solution of a problem by applying operators such as crossover and mutation. It is important to select proper parameter settings and operators for GAs to work efficiently. With previous studies in parameter tuning [25], we decide to perform the GAs on IOHprofiler [7] platform. In terms of the black-box optimization problems on IOHprofiler, different algorithms may be the best suited. Moreover, the static optimal parameter settings may not be the optimal parameter setting in any different optimization stage of a problem sometimes. In this way, the static optimal parameter settings may prevent us from the best configuration. The question arises if adequate settings can be found automatically during the process.

We review relevant studies about self-adaptation, which is given from evolution strategies. Self-adaptation is the implicit search in the space of strategy parameters [19]. The self-adaptive control of mutation strengths in evolution strategies was exceptionally successful. A previous study [12] showed a bimodal function for which the algorithm does not converge to the global optimizer with probability one if it starts close to the local optimal solution. It also indicates that evolution strategies with multi-recombinative self-adaptation strategies show sensitive behaviour concern-

ing the learning rate, which results in not realizing optimal mutation strength. Under simple fitness functions considered, conditions can be derived that ensure the convergence of the EA to local optimal solutions. But for the black-box Pseudo-Boolean Optimization (PBO) problems, we cannot ensure that the self-adaptation could solve this issue because an effective adaptive method also depends on the features of problems and their complex fitness functions.

Despite self-adaptation being unsuitable for our study, we take the concept of changing algorithm configuration into consideration. With the significant development in Machine Learning and in exploratory landscape analysis, it is possible that the dynamic algorithm configuration can be solved by automatically trained configuration schedules. Regarding using Machine Learning to tune the parameters in the process, we consider that reinforcement learning could be a good method. But we don't devise a straightforward way to set up the reward function for reinforcement learning of solving process.

Then, we focus on the configuration selection based on data. Aiming to find the optimal parameter setting for each optimization stage of a problem during the process, we follow the idea of selecting a best-suited algorithm called the dynamic algorithm selection (dynAS) problem. Previous research [23] investigates the potential improvement that can be achieved from switching between solvers. We apply the dynAS on the same problems in [23] which are provided by IOHprofiler. The post empirical analysis is also implemented on IOHprofiler.

In previous study, a way to analyze the computational complexity is the optimization time, which is measured by means of the number of function evaluations. But it is also demonstrated that counting function evaluations more precisely can lead to results contradicting actual run times [16]. We take the number of function evaluations into consideration as a reference Information which is related to number of runs and other stochastic factors such as initialization. Regarding dynAS problems, we evaluate algorithms using expected running time (ERT). As ERT could be easily obtained from IOHprofiler, we use it as an performance measure to analyze GAs. With previous study about dynAS problems based on ERT, we consider that there could be another performance measure which is more efficient and informative. The new measure we attempt to use is area under the ECDF curve (AUC). ECDF curve is the abbreviation of Empirical Cumulative Distribution Function, which shows the proportion of the runs that have found a solution of at least the required target value within the budget given by the $x$-axis.

## 1.3 Outline

The thesis consists of five themed chapters. Chapter One is the introduction that contains relevant studies and how we settle our aims. We review previous studies about parameter configuration on genetic algorithms and dynamic configuration methods. With those concepts in mind, we finally decide to use AUC as a performance measure of dynamic algorithm selection framework to optimize the best configuration at all stages.

Chapter 2 introduced the background of the genetic algorithm. Then it presents the preliminaries of the thesis. It includes the definition of expected running time, which is the key comparison data for algorithms or policies judgement. Then, we introduce the definition of the area under the curve that we use as an index. Besides, we present the essential algorithm framework of $(\mu + \lambda)$ Genetic Algorithm.

The third chapter concerns the methodology used for this thesis, including the parameter settings set in our algorithms, the design of experiments, the calculation method of dynamic indexes ERT and AUC, and the dynamic algorithm selection policies. Chapter 4 analyzes both the theoretical and empirical results of experiments and focus group discussions on the performance of policies. Lastly, we draw conclusions based on our investigations.

Overall, we highlight the use of AUC as a measure for better dynamic algorithm selection on the 25 IOHprofiler benchmark problems. In addition, we summarize the effect of the derivative of AUC on dynAS problems for parameter settings such as population size and switch target. An inspiring conclusion we obtained is that AUC could be used as a measure to perform better than ERT as a measure for many problems. And future work should focus on the way of transformation on switching algorithm and other methods for dynamic parameter selection to solve the limitation of the AUC selection policy.

# Chapter 2

# Background

## 2.1 Genetic Algorithms

In the last century, scientists have been interested in the simulation on computers of evolution. Starting in 1957, Nils Aall Barricelli published his artificial methods on simulation of evolution processes [4]. Other scientists, such as Alex Fraser, published a series of papers on the simulation of artificial selection of organisms [10]. From these beginnings, computer simulation of evolution by biologists became more common in the early 1960s, and the methods were described in books by Fraser and Burnell (1970) and Crosby (1973) [9]. Of that book, the simulations included all of the essential elements of modern genetic algorithms.

With grown up computer simulation, scientists have published a series of papers that also adopted a population of solution to optimization problems, undergoing recombination, mutation, and selection. Regarding the algorithm in application, Barricelli had simulated the evolution of ability to play a simple game [3], leading to the ideas of solving complex engineering problems.

Genetic algorithms in particular became popular through the work of John Holland in the early 1970s, and particularly his book *Adaptation in Natural and Artificial Systems* [13]. It introduced the well-known theory as Holland's Schema Theorem for predicting the quality of the next generation. Researches remained theoretical until the mid-1980s have limitations. We got some criticism on Schema Theorem from Thomas Bäck [1]:

- Most of Holland's approximations are only true for very large numbers (trials and population size)

- Within finite populations, exponentially increasing the number of schema instances leads to entirely filling the population
- Within finite populations, exponentially decreasing the number of schema instances leads to complete elimination
- Not all schemata are represented in a typical population
- Schemata of large defining length are likely to be destroyed by crossover (even highly fit ones)

Nowadays, the genetic algorithms is mature with a stable framework containing **Initialization**, **Selection**, **Genetic operators**, **Heuristics**, and **Termination** [24]. Initialization provides the initial population randomly, allowing the entire range of possible solutions. Selection is the process of choosing individuals with better performance. Individuals with better performance are more likely to be selected as measured by a fitness function. There are plenty of selection methods, such as the most popular roulette wheel selection (also known as fitness proportionate selection), rank selection, steady state selection, etc. To generate the second generation of solutions from those selected, GA uses a combination of genetic operators: crossover and mutation. Heuristics are designed to accelerate and robust the calculation. Heuristic penalizes crossover between similar candidate solutions and helps prevent premature convergence to a local optimal solution [22]. Last but not least, the genetic algorithm process will be executed until the termination condition has been reached.

## 2.2 Performance Measures

In this section, we provide the definitions of two performance measures: expected running time (ERT) and area under the ECDF curve (AUC), respectively. First of all, we denote the algorithm set as $\mathcal{A} = \{A_1, A_2, ..., A_n\}$ which contains $n$ algorithms that may be used in our thesis.

### 2.2.1 Expected Running Time

The expected running time (ERT) shows the performance of an algorithm. The ERT of an algorithm $A$ hitting a target $\phi$ on a problem $P$ is given as below 2.1 [25]:

$$\mathrm{ERT}(A, P, \phi) = \frac{\sum_{i=1}^{r} \min\{t_i(A, P, \phi), B\}}{\sum_{i=1}^{r} \mathbb{1}\{t_i(A, P, \phi) < \infty\}} \tag{2.1}$$

where $r$ is the number of runs of the algorithm $A$, and $B$ is the maximal budget of the algorithm $A$ on the problem $P$. $t_i(A, P, \phi)$ is the running time of algorithm $A$ on problem $P$ to get target $\phi$. Besides, the $\mathbb{1}$ we use to ensure the convergence in the study indicates:

$$\mathbb{1}\{\text{condition}\} = \begin{cases} 1, & \text{if the condition is true} \\ 0, & \text{else} \end{cases} \tag{2.2}$$

There comes the the best static algorithm (BSA) $A^*$ for problem $P$ with target $\phi$ as below. We use the BSA as a benchmark for comparing with the performance of $\pi$ in study.

$$A^* = \arg\min_{A \in \mathcal{A}} \mathrm{ERT}(A, P, \phi) \tag{2.3}$$

### 2.2.2 Area Under the ECDF Curve

Area under the ECDF curve (AUC) can be calculated by different methods, of which the integral method of finding the area is most popular. We follow the definition of AUC [11] and transform it to fit our dynAS problem as follow.

$$\mathrm{AUC}(A, P, \phi) = \frac{\sum_{i=1}^{r} \sum_{j=1}^{m} \sum_{h=1}^{z} \mathbb{1}\{f_P(A, B_j, i) \geq \phi_h\}}{r \cdot m \cdot z} \tag{2.4}$$

Equation (2.4) indicates a way to calculate the AUC value for dynAS problem where $r$ is number of runs. $m$ is the size of budget set $\mathcal{B} = \{B_1, B_2, ..., B_m\}$ where $\max(\mathcal{B}) = B_m$, the largest used budget for problem $P$. $z$ is the size of target set $\Phi = \{\phi_1, \phi_2, ..., \phi_z\}$ where $\max(\Phi) = \phi_z$ is the target maximization that we consider in this thesis. And $f_P(A, B_j, i)$ is the function value of algorithm $A$ on problem $P$ within budget $B_j$ in $i$-th run.

We have the same definition about $\mathbb{1}$ as ERT as Equation (2.2). In this way, the AUC value is restricted in $(0, 1)$, which is fairly dependent on the budget set and target set.

### 2.2.3 A family of $(\mu+\lambda)$ GA

The algorithm framework in this thesis is a family of $(\mu + \lambda)$ Genetic Algorithm, where $\mu$ is the size of parent and $\lambda$ is the size of offspring. In addition, it has crossover probability $p_c$ and mutation rate $p$. It is allowed for us to tune parameters and select from a set of operators in the configurable GA framework. The pseudocode Algorithm

1 describe the framework in detail.

**Algorithm 1:** A Family of $(\mu + \lambda)$ Genetic Algorithm

**Input** : Population sizes $\mu$, $\lambda$, crossover probability $p_c$, mutation rate $p$;

**Termination:** The optimum is found or the budget is used out;

**Initialization**: **for** $i = 1, \ldots, \mu$ **do**
  Sample $x^{(i)} \in \{0,1\}^d$ uniformly at random (u.a.r.), and evaluate $f(x^{(i)})$;
**end**
Set $P = \{x^{(1)}, x^{(2)}, \ldots, x^{(\mu)}\}$;
**Optimization:** **for** $t = 1, 2, 3, \ldots$ **do**
  $P' \leftarrow \emptyset$;
  **for** $i = 1, \ldots, \lambda$ **do**
    Sample $r \in [0,1]$ u.a.r.;
    **if** $r \leq p_c$ **then**
      select two individuals $x, y$ from $P$ u.a.r. (with replacement);
      $z^{(i)} \leftarrow$ Crossover$(x, y)$;
      **if** $z^{(i)} \notin \{x, y\}$ **then**
        evaluate $f(z^{(i)})$;
      **else**
        infer $f(z^{(i)})$ from parent;
      **end**
    **else**
      select an individual $x$ from $P$ u.a.r.;
      $z^{(i)} \leftarrow$ Mutation(x);
      **if** $z^{(i)} \neq x$ **then**
        evaluate $f(z^{(i)})$;
      **else**
        infer $f(z^{(i)})$ from parent;
      **end**
    **end**
    $P' \leftarrow P' \cup \{z^{(i)}\}$;
  **end**
  $P$ is updated by the best $\mu$ individuals in $P \cup P'$;
**end**

The algorithm initializes the population set $P$ with $\mu$ individuals uniformly at

random in $\{0,1\}^d$ where $d$ is the dimension of the dynAS problem. As we mentioned in research questions, we use `Crossover` and `Mutation` operators in the algorithm. There are three ways of crossover and two ways of mutation, which will be detailed in the next chapter. $\lambda$ offspring is generated by either using crossover with probability $p_c$ or using mutation with probability $1 - p_c$. The best $\mu$ individuals of parent and offspring will be selected as the parent for the next iteration. This algorithm terminates when hitting the optimum of function or running out of the maximal budget of function evaluations.

## 2.3 Algorithm Configuration/Selection

With the definitions of ERT, AUC, and the framework of $(\mu + \lambda)$ GA in mind, the definition of the problems we face in the thesis needs to be clarified. A common problem encountered with parametric algorithms is to find parameter configurations where the empirical performance is optimized over a given set of problem instances like the 25 black-box problems. Formally, this *algorithm configuration* or *parameter tuning* problem can be stated as follows [14]:

Given

- an algorithm $A$ with parameters $p_1, \ldots, p_k$ that affect its behavior,
- a space $C$, of configurations, where each configuration $c \in C$ specifies values for $A$'s parameters s.t. $A$'s behaviour on a given instance is completely specified,
- a set of problem instances $I$,
- a performance metric $m$ that measures the performance of $A$ on instance set $I$ for a given configuration $c$,

the algorithm configuration problem aims to find a configuration $c^* \in C$ that results in optimal performance of $A$ on $I$ according to metric $m$.

The algorithm whose performance is to be optimised is often called the target algorithm. Various types of parameters may occur depending on the given target algorithm. With the clear definition of parameter tuning problem, we move steps to the solutions of algorithm configuration problems.

D. Knuth said "Parameter optimization for general broad-spectrum use is a daunt-

ing task, not only because of significant differences between species of SAT instances but also because of the variability due to random choices when solving any specific instance. It's hard to know whether a change of parameter will be beneficial or harmful..." in his book [17]. It is towards the Automatic Algorithm Configuration which applies powerful search techniques to design algorithms, use computation power to explore algorithm design spaces, and free human creativity for higher-level tasks.

Aiming to figure out the target algorithm, scientists came up with many models for Automatic Algorithm Configuration, such as irace, SMAC, ParamILS, and Bayesian optimization, etc. Irace is a variant of I/F-Race (Iterated F-Race), which is designed to sample a set of algorithm configurations by iterative refinement of a sampling model [5] [2], with several extensions. The latest irace is proposed by López-Ibáñez, Dubois-Lacoste and Stützl [18]. Irace is widely common for sequential statistical testing. It samples new configurations according to a probability distribution, then selects the best configurations from the newly sampled ones by means of racing, and finally updates the probability distribution in order to bias the sampling towards the best configurations.

Regarding Sequential model-based algorithm configuration (SMAC) [15], it grasps the idea of using surrogate models to predict performance. SMAC extends surrogate model-based configuration to complex algorithm configuration tasks and across multiple instances. ParamILS is an iterated local search method that operates within the parameter space. It processes initialization by selecting the best configuration among default and several random configurations, then manages a local search: 1-exchange neighbourhood, where exactly one parameter changes a value at a time. Meanwhile, the neighbourhood is searched in random order. After that, the perturbation: change several randomly chosen parameters. The acceptance criterion is selecting the better configuration.

Bayesian optimization, a sequential design strategy for global optimization of black-box functions, is generally attributed to Jonas Mockus and his work [20]. It's more commonly used for statistic problems as the method treats a problem as a random function and sets a prior on it. Priors capture beliefs about the behavior of functions. After the collection of function evaluations processed as data, the previous is updated to form the posterior distribution of the target function. Then the posterior distribution will be used to construct an acquisition function that determines the next function point. Frank Hutter, Holger Hoos, and Kevin Leyton-Brown also applied this method for automatic algorithm configuration in paper [15]. Besides, J. Snoek and H. Larochelle applied the method on parameter tuning for machine learning [21].

### 2.3.1 Dynamic Algorithm Configuration/Selection

Regarding this thesis, we focus on the one-shot dynamic algorithm configuration/selection (dynAS) problem that follows the statements from Holger H. Hoos [14] we mentioned before. First, the genetic algorithm $A$ has parameters $\mu, \lambda, p_c, p$ that affect its behavior. The space $C$ of configurations, where each configuration $c \in C$ specifies values for $A$ will be detailed in the next chapter. We have a set of problem instances $I$. And last but not least, we introduced two performance measures ERT and AUC that measure the performance of $A$ on instance set $I$ for a given configuration $c$.

The one-shot dynamic selection policy is restricted to switching only once during the process. The *dynamic* is reflected in our parameter settings, as we use multiple configurations on an instance that will switch from one to another. It will be detailed in the subsequent experiments chapter3.1. The issues we address in this thesis contain the usage of different performance measures on dynamic configurations and the optimization of instances by tuning the dynamic configuration.

# Chapter 3

# Experiment

In this chapter, we first present the parameter settings in experiments under the $(\mu+\lambda)$ GA framework as algorithm 2. Secondly, we explain the theoretical calculation of ERT and AUC on dynAS problems. In this way, we are able to calculate the ERT and AUC values of the dynAS problem with different policies. And we present our dynamic GA design, including the pseudocode implementation, an instance of the feature of AUC, which indicates the process of reaching the final target of an algorithm in ten runs, and the method of combining algorithms.

## 3.1 Parameter settings of GA framework

In this study, we use crossover and mutation operator under the framework of the GA family algorithm we introduced in Chapter 2.2.3. There are three different crossover operators, including *one-point crossover*, *two-point crossover*, and *uniform crossover*. Besides, the two mutation operators are *standard bit mutation* and *fast mutation*.

Regarding the *standard bit mutation*, we use a binomial distribution $\mathrm{Bin}_{>0}(d, p)$ to decide the number of bits $\ell$ which is called mutation strength to flip. $d$ is the dimension of the problem and $p$ is fixed as $1/d$ in our experiment. $\ell$ bits are randomly chosen from an individual.

For the *fast mutation*, the number $\ell$ is sampled from a power-law distribution

suggestion [6] ($\beta = 2$) as follow:

$$\Pr[X = \alpha] = \left(C_{n/2}^{\beta}\right)^{-1} \alpha^{-\beta} \quad \text{for all } \alpha \in [1, ..., n/2] \tag{3.1}$$

$$C_{n/2}^{\beta} := \sum_{i=1}^{n/2} i^{-\beta} \tag{3.2}$$

The equation (3.1) defines a probability distribution with power-law factor $C_{n/2}^{\beta}$, which depends on the power-law parameter $\beta$. It also constraints that the maximum number of mutation individuals is less than $n/2$ but at least one. The fast mutation provides a higher probability for choosing more individuals for mutation comparing with standard bit mutation. For instance, if we run fast mutation with the $\beta = 2$ and standard mutation 2,000 times individually, the mutation strength *ell* of fast mutation is 4.85 while the standard mutation is 1.56.

For population size, we choose $\lambda$ from $\{1, 10, 50, 100\}$ and we test the offspring size from the four schemata $(\lambda + 1)$, $(\lambda + \lambda/2)$, and $(\lambda + \lambda)$ and $(1 + \lambda)$. Then we have the algorithm set $\mathcal{A}$ containing GAs with parameters listed below:

- 4 population size schemata: $(\lambda + 1)$, $(\lambda + \lambda/2)$, and $(\lambda + \lambda)$ for $\lambda \in \{10, 50, 100\}$, and $(1 + \lambda)$ for $\lambda \in \{1, 10, 50, 100\}$.
- 2 mutation operators: standard bit mutation with $p = 1/d$ and fast muatation with $\beta = 2$.
- 3 crossover operators with $p_c \in \{0, 0.5\}$: one-point crossover, two-point crossover, and uniform crossover

Crossover operators are only applied for $(\lambda + 1)$, $(\lambda + \lambda/2)$, and $(\lambda + \lambda)$ with $p_c$. The other GAs are all mutation-only.

## 3.2 Dynamic calculation in theory

In terms of the dynAS problem, we follow the previous study[25] about one-shot dynamic selection policy $\pi = (A_1, A_2, \phi_s)$, which is restricted to switch only once in this work. In other words, the policy $\pi$ allows the problem switch from using an algorithm $A_1$ to an algorithm $A_2$ at the switch target $\phi_s \in \Phi$ where $\Phi$ is the target set.

### 3.2.1 ERT for dynAS problem

The ERT of a policy $\pi = (A_1, A_2, \phi_s)$ in theory can be constructed as the combination of two algorithms part. Then there comes up the predicted ERT performance of $\pi$ hitting the final target $\phi_f$ on a problem $P$:

$$\mathrm{ERT}\left(\pi, P, \phi_f\right) = \mathrm{ERT}\left(A_1, P, \phi_s\right) + \mathrm{ERT}\left(A_2, P, \phi_f\right) - \mathrm{ERT}\left(A_2, P, \phi_s\right) \tag{3.3}$$

By using the definition of predicted performance on ERT above, the best dynamic algorithm selection policy(BDA) $\pi^*$ under ERT is defined as Equation (3.4).

$$\left(A_1, A_2, \phi_s\right)^* = \pi^* = \underset{\pi \in (\mathcal{A} \times \mathcal{A} \times \Phi)}{\arg\min}\ \mathrm{ERT}\left(\pi, P, \phi_f\right) \tag{3.4}$$

There are advantages using ERT definition to predict and measure a selection policy $\pi$ in dynAS problems. Firstly, the definition is straightforward to understand its meaning, which calculates the expected running cost of a selection policy as below. Then, the prediction of a combination of two algorithms is easy to calculate and verify.

However, the calculation of ERT depends on each run and will be affected by unsuccessful runs. For instance, if problem F1 could find optimum in 8 out of 10 runs, the ERT will be too high to be meaningful. Secondly, ERT could only be produced after a whole loop, which means we could not monitor it through the process of resolving a problem.

### 3.2.2 AUC for dynAS problem

Correspondingly, we consider about the predicted AUC performance of $\pi = (A_1, A_2, \phi_s)$ hitting the final target $\phi_f$ on problem $P$. We define the "theoretical" predicted AUC value of a dynAS policy $\pi$ to find a switch target $\phi_s$ of a given problem P.

$$\begin{aligned}\mathrm{AUC}\left(\pi, P, \phi_f\right) =& \frac{\sum_{i=1}^{r}\sum_{j=1}^{s_B}\sum_{h=1}^{z} \mathbb{1}\{f_P(A_1, B_j, i) \geq \phi_h\}}{r \cdot m \cdot z} \\ &+ \frac{\sum_{i=1}^{r}\sum_{j=s_B+1}^{m}\sum_{h=1}^{z} \mathbb{1}\{f_P(A_2, B_j + \Delta_{B_{\phi_s}}, i) \geq \phi_h\}}{r \cdot m \cdot z}\end{aligned} \tag{3.5}$$

$$\Delta_{B_{\phi_s}} = B_{\phi_s}(A_2) - B_{\phi_s}(A_1) \in [-B_m, B_m] \tag{3.6}$$

$\Phi = \{\phi_1, \phi_2, ..., \phi_z\}$ is a given set of targets of problem P and $\max(\Phi) = \phi_f$. The switch features are presented as $B_{\phi_s}$, which is the used budget of hitting target $\phi_s$.

$B_{\phi_s}(A_2)$ and $B_{\phi_s}(A_1)$ indicate the used budget of algorithm $A_2$ and $A_1$ respectively for reaching target $\phi_s$. So $\Delta_{B_{\phi_s}}$ is the difference of $B_{\phi_s}$ on algorithm $A_1$ and algorithm $A_2$.

Budget set $\mathcal{B} = \{B_1, B_2, ..., B_m\}$ is also decided by problem $P$ with $\max \mathcal{B} = B_m$. $s_B$ denotes the rank of $B_{\phi_s}$ in $\mathcal{B}$, which can be calculated in each run from settled switching target $\phi_s$. The $f_P(A_1, B_j, i)$ is the maximum value of fitness function in run $r$ within budget $B_j$ using algorithm $A_1$.

By using the definition of predicted performance on AUC above, the best dynamic algorithm selection policy $\pi^*$ under AUC is defined as Equation (3.7).

$$(A_1, A_2, \phi_s)^* = \pi^* = \underset{\pi \in (\mathcal{A} \times \mathcal{A} \times \Phi)}{\arg\min} \operatorname{AUC}(\pi, P, \phi_f) \tag{3.7}$$

Selecting AUC value as a performance measure to reflect the performance of dynamic parameter settings has advantages. Firstly, it can be calculated after each evaluation instead of a whole loop as ERT. Secondly, the derivative of AUC reflects the trend of GAs which helps select policies. Besides, ERT values can be significantly influenced by the number of unsuccessful runs, resulting in extremely large values. However, we don't have this problem using AUC as the performance measure.

On the other hand, we have to tune more settings for AUC, including a proper budget set and a target set. The two sets have a significant impact on AUC value if the two sets are small. Moreover, the difference in AUC values between algorithms is not large enough to recognize performance as they are all in $[0, 1)$. So we combine the two performance measures for dynAS problems in the following context.

## 3.3 Settings of dynamic policies

In this section, we present the details of the implementation of dynamic GA in our experiment, including the pseudocode and the transformation from one parameter setting to another. And we present the way of choosing the combinations of all parameter settings.

### 3.3.1 Methods for implementation of dynamic GA

Similar to the GA framework in Sec. 2.2.3, our dynamic GA framework also have the parameters, including 4 population size schemata, 2 kinds of mutation operators, and 3 crossover operators with $p_c \in \{0, 0.5\}$. The difference is that the parameters are

double as there are two GAs to be applied. The pseudocode Algorithm 2 describes the framework in detail.

The transformation process in Algorithm 2 from lines 31 to 40 explains our method of changing population size. On one hand, duplicating individuals from $A_1$ guarantees a lower bound which is equal to or bigger than $\phi_s$. On the other hand, updating half of $P$ by mutation ensures population diversity. Otherwise, we only obtained the same individuals from duplicating, which is meaningless to crossover. Correspondingly, for the case that the population size of $A_2$ is smaller than $A_1$, we select the best $\mu$ individuals from $P$ as new parents. We implement different types of transformation in an experiment, which is detailed in Chapter 4.

### 3.3.2 Features of AUC

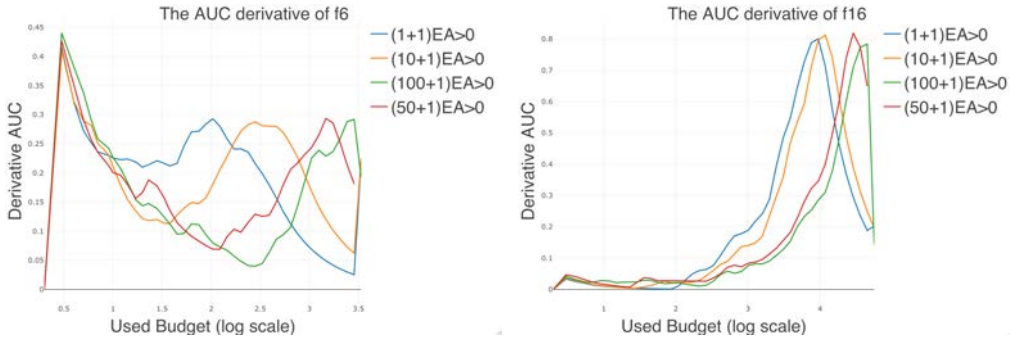
As we introduced before, AUC value is related to the target set $\Phi$ and budget set $\mathcal{B}$. We are interested in the AUC and its feature on reflecting the extent to which the target has been met. Assuming that we have a determined target set and budget set in hand, we are allowed to calculate the AUC value during the process of solving problems.

We calculate the AUC value of the 25 PBO problems from the IOHprofiler with different parameter settings to check their similarities and difference. We generate the figures of the derivative of AUC with respect to the budget of those problems. In terms of different parameter settings, the main category is their population size, which we choose $\{1, 10, 50, 100\}$ for analysis. Figure 3.1 contains two images about the derivative of AUC from F6 and F16 of the IOHprofiler.

The reason for choosing F6 and F16 is that they are in different kinds of problems. F6 and F16 present Neutrality and Fitness perturbation W-model transformations to the OneMax and LeadingOnes respectively[8]. With these two functions, we have a brief impression of the AUC features that it could suit to reflect the efficiency of different types of functions.

The lines in blue are the population size $\{\mu = 1\}$, the lines in orange are $\{\mu = 10\}$, the lines in green are $\{\mu = 50\}$, and the lines in red are $\{\mu = 100\}$. There's a common feature among the two images that the first peaks of the lines are early and overlapped, which is related to the the initialization.

The second peak of each line depends on their population size: $\{\mu = 1\}$ comes first, then followed by $\{\mu = 10\}$, $\{\mu = 50\}$, and $\{\mu = 100\}$. The reason is that the algorithms with smaller population sizes are easier to make progress at the beginning




**(a)** *AUC derivative of GAs on F6* **(b)** *AUC derivative of GAs on F16*

**Figure 3.1: AUC derivative of GAs on F6 and F16 in dimension $d = 100$**

stage.

But on the other hand, the algorithms with small population sizes could also fall earlier. The decline indicates the speed of an algorithm making progress on a problem is getting slower. The area under the lines indicates the AUC value, which we aim to maximize, could not be compared directly. In this way, the dynAS policy should be selected in a numerical way which is presented in the following section. With this feature we summarize from the figures, we focus on the combination of different parameter settings, aiming to get a higher AUC value using less budget.

### 3.3.3 Combination and selection method

With the features of AUC in mind, we take more functions into consideration. We select 10 proper policies for each problem $\pi$ among all possible combinations of algorithms. Assuming that we have 20 potential switch targets for each problem, there are $\binom{80}{2} \times 20 = 126400$ combinations for each problem as we have 80 GA algorithms.

With the target set $\Phi$ and budget set $B$ for calculating the AUC values reflecting the effect of parameter settings for each problem, we expect to gain improvement from the second algorithm $A_2$ building on the previous algorithm $A_1$. By using the definition of dynamic AUC (Equation 3.5), we calculate the dynamic AUC in theory, of which the result is presented in Sec. 4.1.

For the implementation of AUC calculation, we first pick a pair of algorithms among all combinations. Then we pick the potential switch targets one by one to form a policy $\pi = (A_1, A_2, \phi_s)$. On the one hand, Regarding 25 problems from IOHprofiler respectively, the 80 algorithms cannot guarantee to find the optimum target in each

run. On the other hand, the enumeration method for combination is not efficient enough to save running time.

So we add a filter to drop the algorithms which could not find optimum within the budget $B_m$ of BSA. Because we want an improvement based on BSA using dynamic policy. With the sorted results by AUC and budget from the combination process, we obtain the top 10 selection policy $\pi_i, i = 1, ..., 10$ in theory. And the best result for each problem in theory, is detailed in Sec. 4.1. To validate the theoretical values, we use the dynamic GA 2 to run the top 10 policies from AUC calculation for each problem.

**Algorithm 2: Dynamic GA for policy $\pi = (A_1, A_2, \phi_s)$**

**Input** : Population sizes $\mu^{(1)}$, $\lambda^{(1)}$, crossover probability $p_c^{(1)}$ for algorithm $A_1$; Population sizes $\mu^{(2)}$, $\lambda^{(2)}$, crossover probability $p_c^{(2)}$ for algorithm $A_2$; mutation rate $p$, switch target $\phi_s$;

**Termination:** The optimum is found or the budget is used out;

**Initialization**: **for** $i = 1, \ldots, \mu^{(1)}$ **do**
  Sample $x^{(i)} \in \{0,1\}^d$ uniformly at random (u.a.r.), and evaluate $f(x^{(i)})$;
**end**
Set $P = \{x^{(1)}, x^{(2)}, \ldots, x^{(\mu)}\}$;
Set $f_{opt} = -\infty, p_c = p_c^{(1)}, \lambda = \lambda^{(1)}$;
**Optimization: for** $t = 1, 2, 3, \ldots$ **do**
  $P' \leftarrow \emptyset$;
  **for** $i = 1, \ldots, \lambda$ **do**
    Sample $r \in [0,1]$ u.a.r.;
    **if** $r \leq p_c$ **then**
      select two individuals $x, y$ from $P$ u.a.r. (with replacement);
      $z^{(i)} \leftarrow$ `Crossover`$(x, y)$;
      **if** $z^{(i)} \notin \{x, y\}$ **then**
        evaluate $f(z^{(i)})$;
      **else**
        infer $f(z^{(i)})$ from parent;
      **end**
    **else**
      select an individual $x$ from $P$ u.a.r.;
      $z^{(i)} \leftarrow$ `Mutation`$(x)$;
      **if** $z^{(i)} \neq x$ **then**
        evaluate $f(z^{(i)})$;
      **else**
        infer $f(z^{(i)})$ from parent;
      **end**
    **end**
    $P' \leftarrow P' \cup \{z^{(i)}\}$;
    `Transformation`$(f_{opt}, \phi_s, P)$;
  **end**
  $P$ is updated by the best $\mu$ individuals in $P \cup P'$;
  $f_{opt}$ is updated by the biggest function value in $P$;
  **Transformation: if** $f_{opt} < \phi_s$ **then**
    break;
  **else**
    $p_c = p_c^{(2)}, \mu = \mu^{(2)}, \lambda = \lambda^{(2)}$;
    **if** $\mu^{(2)} \geq \mu^{(1)}$ **then**
      Enlarge $P$ by duplicating $\mu^{(1)}$ individuals of $P$ to size $\mu^{(2)}$;
      Update $\mu^{(2)}/2$ individuals of $P$ by mutation randomly;
    **else**
      $P$ is updated by the best $\mu^{(2)}$ individuals in $P$;
    **end**
  **end**
**end**

# Chapter 4

# Results

This chapter aims to present the results of our study, including the theoretical results and empirical results. To distinguish between the policies which are generated with AUC as a performance measure and the BSA under standard $(\mu + \lambda)$ GA framework, we run the experiments. First of all, we calculated the theoretical AUC values to decide the policies for each problem. Then, we used those policies to solve the 25 problems and got the empirical results.

In addition, we also aim to find out the difference between policies with ERT and AUC as performance measures. Regarding the policies, we refer to the previous study [25] where ERT was used as a measure for comparison. Considering the difference between the two types of selection policy, we notice that the presented result differs on the switch target and the algorithms involved.

Table 4.1 indicates the theoretical results of ERT selection policy that we make comparisons to. *funcId* is the column of ids of problems from IOHprofiler, *fTarget* indicates the final targets of different functions. *sTarget* is the switch target that the algorithm will change once it is reached. Lastly, *dERT* is the dynamic ERT value in theory, calculated based on the equation 3.3 and 3.4.

## 4.1 Theoretical results

As the target set $\Phi$ is different but has the same size for each problem, so we use $\{0, 1, ..., 19\}$ to denote the position of switch target $\phi_s$ in $\Phi$. Table 4.2 indicates the policies with the biggest AUC value for each function where $s(\phi_s)$ is the position of switch target $\phi_s$ in $\Phi$ we mentioned. There are 25 policies in this table but we actually

| funcId | fTarget | A1 | A2 | sTarget | dERT |
|---|---|---|---|---|---|
| 1 | 100 | (1+1)EA$_{>0}$ | (10+10)-uniform-GA | 96 | 638 |
| 2 | 100 | (1+1)fastGA | (1+1)EA$_{>0}$ | 21 | 5 186 |
| 3 | 5 050 | (100+1)-two-point-fGA | (1+1)EA$_{>0}$ | 2 899 | 693 |
| 4 | 50 | (1+1)EA$_{>0}$ | (1+10)EA$_{>0}$ | 49 | 379 |
| 5 | 90 | (1+1)fastGA | (1+1)EA$_{>0}$ | 55 | 559 |
| 6 | 33 | (100+100)EA$_{>0}$ | (1+1)EA$_{>0}$ | 20 | 255 |
| 7 | 100 | (100+100)-two-point-GA | (100+50)EA$_{>0}$ | 95 | 182 271 |
| 8 | 51 | (10+10)-uniform-GA | [50+50)-uniform-GA | 50 | 1 441 |
| 9 | 100 | (50+1)-uniform-fGA | (100+100)-uniform-fGA | 96 | 2 255 |
| 10 | 100 | (50+25)-uniform-fGA | (100+100)-uniform-fGA | 94 | 16 956 |
| 11 | 50 | (1+1)fastGA | (1+1)EA$_{>0}$ | 17 | 1 818 |
| 12 | 90 | (1+1)fastGA | (1+1)EA$_{>0}$ | 19 | 4 535 |
| 13 | 33 | (1+1)fastGA | (1+1)EA$_{>0}$ | 14 | 929 |
| 14 | 7 | (100+50)-one-point-GA | (50+25)-uniform-fGA | 5 | 145 |
| 15 | 51 | (1+1)fastGA | (1+1)EA$_{>0}$ | 10 | 6 200 |
| 16 | 100 | (1+1)fastGA | (1+1)EA$_{>0}$ | 21 | 9 455 |
| 17 | 100 | (1+1)fastGA | (1+1)EA$_{>0}$ | 21 | 40 350 |
| 18 | 4.22 | (10+10)EA$_{>0}$ | (50+50)fastGA | 3.57 | 14 468 |
| 19 | 98 | (1+1)fastGA | (1+1)EA$_{>0}$ | 60 | 10 044 |
| 20 | 180 | (1+1)EA$_{>0}$ | (1+10)EA$_{>0}$ | 178 | 1 482 |
| 21 | 260 | (1+1)EA$_{>0}$ | (1+10)EA$_{>0}$ | 258 | 1 041 |
| 22 | 42 | (1+10)EA$_{>0}$ | (100+1)-two-point-GA | 39 | 1 092 |
| 23 | 9 | (1+1)EA$_{>0}$ | (10+1)EA$_{>0}$ | 8 | 1 648 |
| 24 | 17.20 | (1+1)EA$_{>0}$ | 100+100)-two-point-fGA | 15.81 | 1 607 |
| 25 | -0.30 | (1+1)fastGA | (100+100)-uniform-fGA | -0.32 | 11 151 |

**Table 4.1: The theoretical selection policies with lowest ERT value for problems in dimension $d = 100$.**

take 10 policies in consideration for a problem because the top 10 policies have AUC values that are not much different.

The names of algorithms in $A_1$ and $A_2$ columns represent the parameters and operators used therein: 'EA$_{>0}$' denotes the mutation-only GAs using $p = 1/d$, 'fastGA' denotes the mutation-only GAs using fast mutation. '($\mu + \lambda$)-*crossover operator*-GA/fGA' denotes that the GAs run with $p_c = 0.5$, population size $\mu$, and offspring size $\lambda$, of which GA means standard mutation operator using $p = 1/d$ while fGA indicates the fast mutation operator with $\beta = 2$.

As we know that the AUC value will not exceed 1 based on equation (2.4), a higher AUC value indicates a faster process of policy $\pi$ to find the optimal target. Table 4.2 indicates that all the problems could find $\phi_{\max}$ within less budget than the present BSA. Otherwise, the AUC value will not be calculated. Because the budget set is limited with an upper bound as $B_m$, which is the used budget of BSA. On the one hand, the range of AUC values for different problems is also different. Such as F11, F12, and F13, there are functions that find optimum at a late position of the budget set, which leads to a smaller AUC value. F16 has the lowest AUC value 0.6169, which means that the policy $\pi = (\texttt{(1+1)fastGA}, \texttt{(1+1)EA>0}, 27.1)$ takes more time to

| funcId | A1 | A2 | $\phi_s$ | AUC |
|---|---|---|---|---|
| 1 | (50+1)-uniform-fGA | (1+1)EA$_{>0}$ | 55.8 | 0.8437 |
| 2 | (1+1)fastGA | (1+1)EA$_{>0}$ | 31.62 | 0.6288 |
| 3 | (100+1)-one-point-GA | (1+1)EA$_{>0}$ | 2930 | 0.8628 |
| 4 | (100+100)-one-point-fGA | (1+1)fastGA | 27.12 | 0.8229 |
| 5 | (10+5)-uniform-GA | (1+1)EA$_{>0}$ | 50.56 | 0.8128 |
| 6 | (10+5)-two-point-GA | (1+1)EA$_{>0}$ | 20 | 0.8234 |
| 7 | (100+50)-one-point-GA | (50+50)EA$_{>0}$ | 93.44 | 0.9376 |
| 8 | (1+1)EA$_{>0}$ | (10+1)-uniform-GA | 42.66 | 0.8365 |
| 9 | (10+1)-uniform-fGA | (50+50)-uniform-GA | 86.2 | 0.8655 |
| 10 | (50+1)-uniform-fGA | (100+1)-uniform-fGA | 93.31 | 0.9681 |
| 11 | (1+10)fastGA | (1+50)EA$_{>0}$ | 23.76 | 0.6411 |
| 12 | (1+1)fastGA | (1+1)EA$_{>0}$ | 47.4 | 0.6654 |
| 13 | (100+50)-uniform-fGA | (1+1)EA$_{>0}$ | 5.22 | 0.6347 |
| 14 | (100+1)fastGA | (100+1)-two-point-fGA | 4.07 | 0.7095 |
| 15 | (1+10)fastGA | (1+1)EA$_{>0}$ | 11.56 | 0.6385 |
| 16 | (1+1)fastGA | (1+1)EA$_{>0}$ | 27.1 | 0.6169 |
| 17 | (1+1)fastGA | (1+1)EA$_{>0}$ | 37.54 | 0.6545 |
| 18 | (10+10)-two-point-GA | (50+50)EA$_{>0}$ | 3.63 | 0.9156 |
| 19 | (1+1)EA$_{>0}$ | (1+10)EA$_{>0}$ | 88.32 | 0.9265 |
| 20 | (100+1)-one-point-GA | (1+1)EA$_{>0}$ | 108.96 | 0.8583 |
| 21 | (100+100)-one-point-GA | (1+1)EA$_{>0}$ | 165.7 | 0.8522 |
| 22 | (10+1)-uniform-GA | (1+1)fastGA | -2530.8 | 0.9993 |
| 23 | (10+1)-two-point-fGA | (1+1)EA$_{>0}$ | -1259 | 0.9622 |
| 24 | (1+1)EA$_{>0}$ | (100+100)-one-point-fGA | 15.78 | 0.9529 |
| 25 | (1+1)EA$_{>0}$ | (10+5)EA$_{>0}$ | -0.32 | 0.9483 |

**Table 4.2: The theoretical selection policies with biggest AUC value for problems in dimension $d = 100$.**

converge to the optimal target.

On the other hand, F22 has the highest AUC value 0.9993 from the table, which means it could find the optimum within a few budget using the policy $\pi$, where $A_1 =$ `(10+1)-uniform-GA`, $A_2 =$ `(1+1)fastGA`. But as we mentioned above, we also take other policies with smaller AUC value in consideration. Then, we aim to get the experimental performance of the policies that are not shown in the table with dynamic GA in the next section.

## 4.2 Experimental results

Based on the theoretical results above, we apply those policies on the 25 problems from IOHprofiler respectively and got the empirical results. Table 4.3 indicates the best policy $\pi^* = (A_1, A_2, \phi_s)$ from top 10 theoretical results of each problem and its

improvement as ratio(%).

Following previous study [25], we also use ERTs to compare the empirical results because ERTs will not be impacted by the target set and budget set, which would be changed during the experiments. In addition, the policies that generated with ERT as a measure are also used, helping to make clear and direct comparisons.

*fTarget* lists the final target used to calculate ERTs. *sERT* lists the static ERTs from BSA which is also listed in the table. $A_1, A_2$ and *sTarget* list the three kinds of parameters from policies $\pi$. *dERT* lists the ERTs with dynamic algorithm policy $\pi$, and *ratio(%)* lists the difference ratio between dERT and sERT.

$$ratio(\%) = \frac{dERT - sERT}{dERT} \times 100\% \tag{4.1}$$

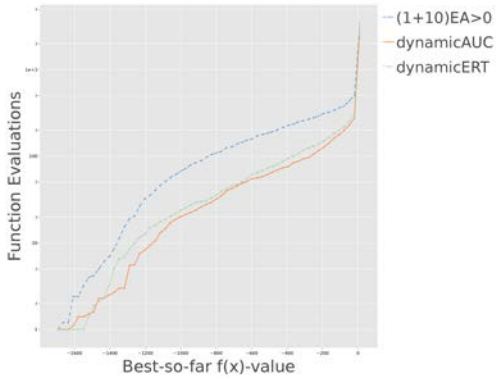
As previously stated, the best policy may not be the top from theoretical results such as F1, which use `(50+1)fastGA` instead of `(50+1)-uniform-fGA` as $A_1$. In addition, From the table, we notice that the ratios of the results are not all positive: F9, F10, F14, F18, F24, and F25 even have much worse dynamic ERT comparing with static ERT of BSA. So we analyze the results to find out the reasons behind.

| funcId | fTarget | BSA | sERT | A1 | A2 | sTarget | dERT | ratio(%) |
|---|---|---|---|---|---|---|---|---|
| 1 | 100 | (1+1)EA$_{>0}$ | 729 | (50+1)fastGA | (1+1)EA$_{>0}$ | 55.8 | 624 | 14.4 |
| 2 | 100 | (1+1)EA$_{>0}$ | 5629 | (10+10)fastGA | (1+1)EA$_{>0}$ | 10.5 | 5093 | 9.5 |
| 3 | 5050 | (1+1)EA$_{>0}$ | 659 | (1+1)fastGA | (1+1)EA$_{>0}$ | 3082 | 674 | -2.3 |
| 4 | 50 | (1+1)EA$_{>0}$ | 414 | (1+1)fastGA | (1+1)EA$_{>0}$ | 28.6 | 401 | 3.1 |
| 5 | 90 | (1+1)EA$_{>0}$ | 593 | (50+1)-two-point-fGA | (1+1)EA$_{>0}$ | 50.5 | 538 | 9.3 |
| 6 | 33 | (1+1)EA$_{>0}$ | 239 | (10+5)-two-point-fGA | (1+1)EA$_{>0}$ | 20 | 264 | -10.5 |
| 7 | 100 | (50+50)EA$_{>0}$ | 143567 | (100+50)-one-point-GA | (50+50)EA$_{>0}$ | 93.4 | 108574 | 24.4 |
| 8 | 51 | (10+1)-uniform-GA | 1083 | (1+1)fastGA | (10+1)-uniform-GA | 37.5 | 867 | 19.9 |
| 9 | 100 | (50+50)-uniform-GA | 2268 | (50+1)-uniform-fGA | (50+50)-uniform-GA | 89.6 | 4144 | -82.7 |
| 10 | 100 | (100+50)-uniform-fGA | 69638 | (50+25)-uniform-fGA | (100+1)-uniform-fGA | 76.1 | 154298 | -121.6 |
| 11 | 50 | (1+1)fastGA | 2096 | (1+10)fastGA | (1+10)EA$_{>0}$ | 23.6 | 1632 | 22.1 |
| 12 | 90 | (1+10)EA$_{>0}$ | 4823 | (1+1)fastGA | (1+10)EA$_{>0}$ | 47.4 | 4737 | 1.8 |
| 13 | 33 | (1+1)EA$_{>0}$ | 1056 | (100+50)-uniform-fGA | (1+1)EA$_{>0}$ | 5.2 | 806 | 23.7 |
| 14 | 7 | (100+1)-one-point-fGA | 55 | (100+1)EA$_{>0}$ | (100+1)-one-point-GA | 5.1 | 112 | -103.6 |
| 15 | 51 | (1+1)EA$_{>0}$ | 6413 | (100+50)-two-point-fGA | (1+1)EA$_{>0}$ | 3.6 | 6573 | -2.5 |
| 16 | 100 | (1+1)EA$_{>0}$ | 9313 | (100+50)-one-point-fGA | (1+1)EA$_{>0}$ | 6.2 | 9025 | 3.1 |
| 17 | 100 | (1+1)EA$_{>0}$ | 41953 | (50+25)-two-point-fGA | (1+1)EA$_{>0}$ | 6.2 | 35309 | 15.8 |
| 18 | 4.22 | (50+50)EA$_{>0}$ | 10432 | (10+1)EA$_{>0}$ | (100+1)EA$_{>0}$ | 3.5 | 12026 | -15.3 |
| 19 | 98 | (1+1)EA$_{>0}$ | 7916 | (50+1)-two-point-fGA | (1+1)EA$_{>0}$ | 55.6 | 8811 | -11.3 |
| 20 | 180 | (1+1)EA$_{>0}$ | 1343 | (100+1)-one-point-GA | (1+1)EA$_{>0}$ | 108.9 | 803 | 40.2 |
| 21 | 260 | (1+1)EA$_{>0}$ | 1129 | (50+50)-one-point-GA | (1+1)EA$_{>0}$ | 165.7 | 731 | 35.3 |
| 22 | 42 | (50+25)-uniform-GA | 2694 | (10+1)-uniform-GA | (1+1)fastGA | -2530.8 | 651 | 75.8 |
| 23 | 9 | (1+10)EA$_{>0}$ | 2505 | (10+1)-two-point-fGA | (1+1)EA$_{>0}$ | -1259 | 2485 | 0.8 |
| 24 | 17.2 | (100+1)-two-point-fGA | 3394 | (1+1)EA$_{>0}$ | (100+1)-two-point-fGA | 15.7 | 500000 | -14631.9 |
| 25 | -0.3 | (10+10)-one-point-fGA | 6556 | (1+1)EA$_{>0}$ | (10+1)EA$_{>0}$ | -0.32 | 13031 | -98.8 |

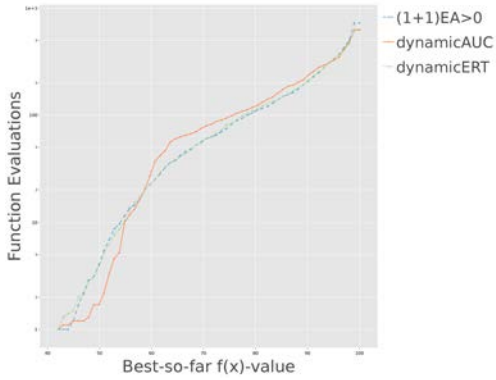
**Table 4.3: The experiment ERT results with BSA and dynamic policies $\pi = (A_1, A_2, \phi_s)$ for 25 problems in dimension $d = 100$.**

### 4.2.1 Earlier switch point can be beneficial

From the empirical experiment data, it represent that some policies are improved relative to BSA as the ratio is positive. Not only the BSA and AUC selection policy we make comparisons, but the ERT selection policy from previous work. Under the same performance measure ERT, the AUC selection policy and ERT selection policy could have a distinguished image. To make the contrast more varied, we select 4 problems with different levels: F23(0.8%), F1(14.4%), F20(40%), and F22(75.8%).




**(a)** *Fixed-target ERT of GAs on F23*




**(b)** *Fixed-target ERT of GAs on F1*

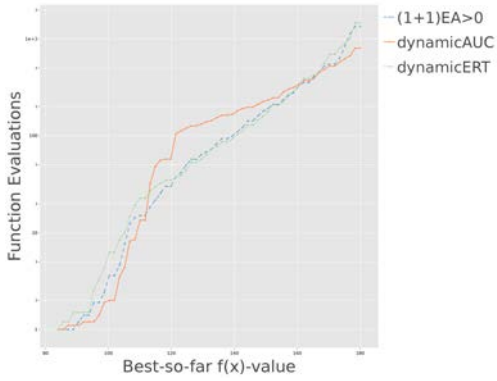
**Figure 4.1: Fixed-target ERTs of GAs on F23 and F1 in dimension $d = 100$. The dynamicAUC in figure 4.1a switches from the $(10+1)$-two-point-fGA to the $(1+1)\mathbf{EA}_{>0}$ at the target $\phi_s = -1259$. The dynamicAUC in figure 4.1b switches from the $(1+1)$fastGA to the $(1+1)\mathbf{EA}_{>0}$ at the target $\phi_s = 55.8$.**

Figure 4.1a and 4.1b present the ERT curves of F23 and F1 respectively. The label *dynamicAUC* and *dynamicERT* indicate the policies that are chosen based on formula (3.4) and (3.7), considering AUC and ERT as cost metric respectively. And the lines in blue stand for the BSAs of the two functions, including `(1+10)EA>0` and `(1+1)EA>0`. The x-axis represents the best function value that has been reached so far, and the y-axis represents the evaluation times of the function during the process.

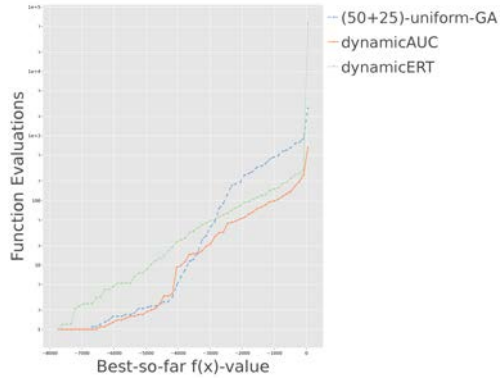
Regarding F23, we notice that the performances of the two dynamic policies are similar. DynamicERT and dynamicAUC are more efficient than the BSA as the two lines are under the blue one before $f(x) = 0$. With the same best-so-far $f(x)$ value, the evaluation of dynamicAUC is less than BSA the whole way through before $f(x) = 0$. Interestingly, dynamicAUC and dynamicERT show almost identical curve trends. It indicates that BSA is not the algorithm that performs the best through the whole pro-

cess. DynamicAUC switches at $\phi_s = -1259$ from `(10+1)two-point-fGA` to `(1+1)EA>0` while dynamicERT switches at $\phi_s = 8$ from `(1+1)EA>0` to `(10+1)EA>0`.

For F1, which has a 14.4% improvement, we also compare it with the performance of BSA. DynamicERT and BSA present similar convergence process before the switch target, while dynamicAUC is not more efficient as it is above the other two lines until the function reaches about 95. DynamicAUC switches at $\phi_s = 55.8$ from `(50+1)fastGA` to `(1+1)EA>0` while dynamicERT switches at $\phi_s = 96$ from `(1+1)EA>0` to `(10+10)-uniform-fGA`. With a random initialization, dynamicAUC can get about 50 at an early stage. A possible reason for the small improvement is that the BSA `(1+1)EA>0` is good enough for this OneMax problem.




**(a)** *Fixed-target ERT of GAs on F20*




**(b)** *Fixed-target ERT of GAs on F22*

**Figure 4.2: Fixed-target ERTs of GAs on F20 and F22 in dimension $d = 100$. The dynamicAUC in figure 4.2a switches from the $(100+1)$-one-point-GA to the $(1+1)$EA$_{>0}$ at the target $\phi_s = 108.9$. The dynamicAUC in figure 4.2b switches from the $(10+1)$-uniform-GA to the $(1+1)$fastGA at the target $\phi_s = -2530.8$.**

Then we focus on the other two functions. Figure 4.2a and 4.2b present the ERT lines of F20 and F22 respectively. The labels are similar to the ERT figures above, including dynamicAUC, dynamicERT and their BSAs.

For F20, dynamicAUC has a 40.2% improvement ratio, while dynamicERT has a better performance before its switch target but fails to get an improvement. DynamicAUC has a point of change close to $f(x) = 120$ in gradient, after which the curve is more efficient than the others. The change point is close to the $\phi_s$, after which the policy uses $A_2$ to solve the function, demonstrating the effectiveness of dynamicAUC policy. DynamicAUC switches at $\phi_s = 108.9$ from `(100+1)-one-point-GA` to `(1+1)EA>0` while dynamicERT switches at $\phi_s = 178$ from `(10+1)-uniform-GA` to

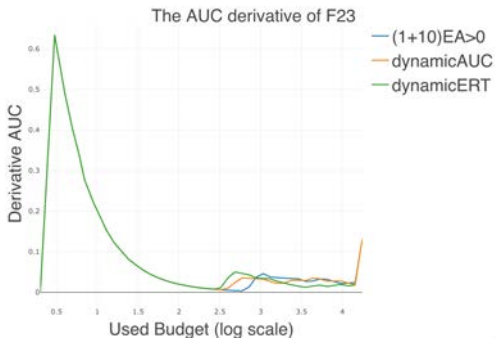
`(1+1)fastGA`.

For F22 with a 75.8% improvement, we also compare it with the performance of BSA. We notice that dynamicAUC performs better than dynamicERT in the whole process because dynamicAUC use fewer evaluations under the same $f(x)$ value. BSA of F22 performs well at first but became harder to make progress gradually. DynamicERT on F22 has a good performance compared with BSA but fails before reaching the final function value target 42, which is related to the late switch. DynamicAUC switches at $\phi_s = -2530.8$ from `(10+1)-uniform-GA` to `(1+1)fastGA` while dynamicERT switches at $\phi_s = 39$ from `(1+10)EA>0` to `(100+1)-two-point-GA`.

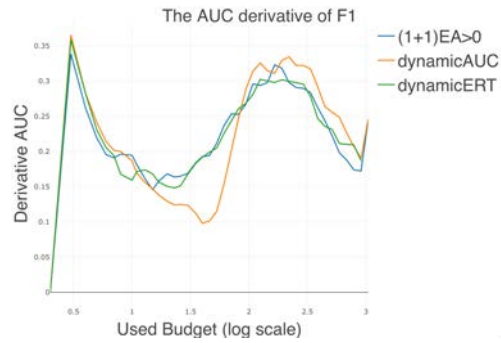
We consider that dynamicAUC policy succeeds in obtaining smaller ERT compared with dynamicERT policy and BSA from the figures above. Not only may the two algorithms in a policy be different, but also dynamicAUC would switch to $A_2$ earlier than dynamicERT.

With the images of ERTs in mind, we focus on the changes in AUC values during the process. We calculate the derivative of AUC on F23 and F1 to check their AUC value trend. Figure 4.3a and 4.3b present the AUC derivative of AUC on F23 and F1 respectively. The curves' colours are similar to ERT figures, where blue is for BSA, orange is for dynamicAUC, and green is for dynamicERT. The x-axis is the used budget during the process with a log operator and the y-axis is the value of derivative AUC.

Regarding F23, figure 4.3a indicates that the increments in AUC value of the three GAs are similar, which validates the 0.8% improvement and the similar trend in ERT figure 4.1a above.



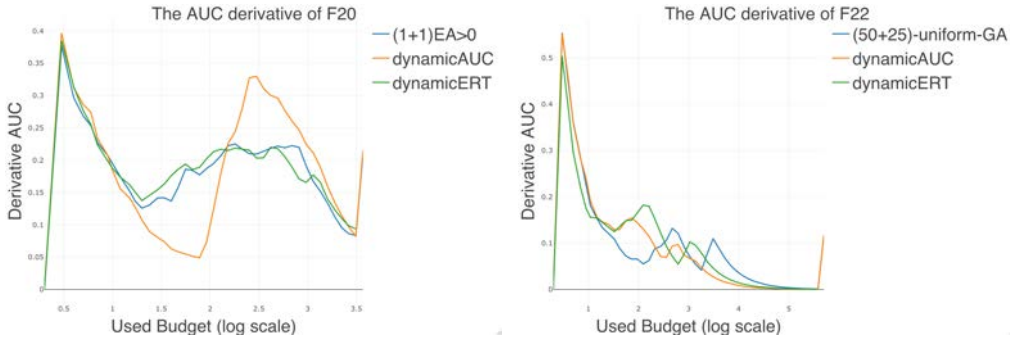

**(a)** *AUC derivative of GAs on F23*




**(b)** *AUC derivative of GAs on F1*

**Figure 4.3: AUC derivative of GAS on F23 and F1 in dimension $d = 100$.**

Different from F23, dynamicAUC on F1 is lower than the other two algorithms during about 1 to 1.8 of the x-axis, which reflects a sooner rise of ERT in figure 4.1b. It is caused by the fact that a lower AUC derivative reflects a higher consumption of budget or evaluations. But dynamicAUC becomes higher on the AUC derivative from about $10^{1.9} = 79.4$ evaluations to $10^{2.9} = 794.3$ evaluations, presenting a lower gradient in the ERT figure of F1. Besides, in the outperform period of dynamicAUC, the best-so-far $f(x)$ value increases from 92 to 100, leading to a 14.4% improvement finally.




**(a)** *AUC derivative of GAs on F20* **(b)** *AUC derivative of GAs on F22*

**Figure 4.4: AUC derivative of GAS on F20 and F22 in dimension $d = 100$.**

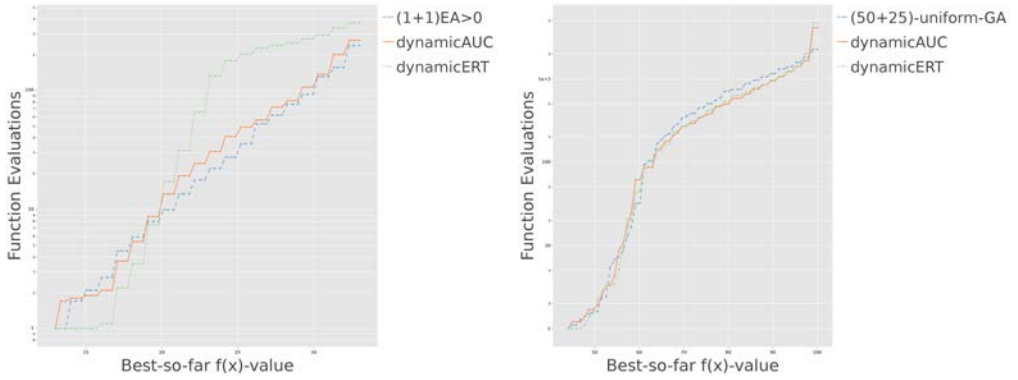
Then, we take the AUC derivative of F20 and F22 into consideration. We find that the AUC derivative of dynmaicAUC on F20 surpasses the other two GAs from evaluation $10^{2.18} = 151$ in figure 4.4a, which matches the point that the dynamicAUC on figure 4.2a changes to a low gradient. It can be seen that the $\phi_s$ of dynamicERT is too late to progress on efficiency.

In contrast, figure 4.4b presents the AUC derivative of F22. It indicates that dynamicAUC has a higher derivative at the beginning, leading to a lower gradient in figure 4.2b. Finally, dynamicAUC reaches 75.8% improvement ratio on F22 with target $f(x) = 42$. Combining with the $\phi_s$ of dynamicERT, we notice that the corresponding evaluation is about $650 = 10^{2.81}$, where the AUC derivative declines to a local minimum. It indicates that the transformation of dynamicERT to $A_2$ make little progress in reaching higher targets. While dynamicAUC switch to its $A_2$ at about evaluation $40 = 10^{1.6}$, where is the local maximum of the AUC derivative, indicating progress in reaching higher targets.

**Summary** We find that comparing with dynamicERT and BSA, dynamicAUC has an earlier switch point $\phi_s$ which helps it combine the advantages of $A_1$ and $A_2$ to get a smaller ERT value. Besides, the derivative of AUC reflects the efficiency of algorithms reaching targets. The figures of AUC derivatives also validate the changes in ERT figures, that AUC selection policies make the process efficient after their switches.

### 4.2.2 Adapting population size is challenging

On the one hand, we obtain some good performances above. On the other hand, we also obtain some bad performances as follows. We select four problems with different levels for various contrast: F6(-10.5%), F9(-82.7%), F14(-103.6%), and F24(not met final target). Not only the BSA and AUC selection policy we make comparisons, but the ERT selection policy from previous work mentioned before.



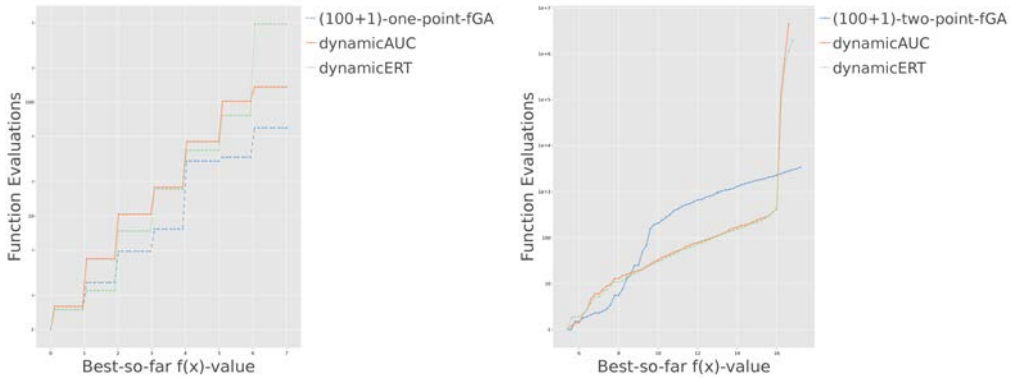

**(a)** *Fixed-target ERT of GAs on F6* **(b)** *Fixed-target ERT of GAs on F9*

**Figure 4.5: Fixed-target ERTs of GAs on F6 and F9 in dimension $d = 100$. The dynamicAUC in figure 4.5a switches from the $(10 + 5)$-two-point-fGA to the $(1 + 1)$EA$_{>0}$ at the target $\phi_s = 20$. The dynamicAUC in figure 4.5b switches from the $(50 + 1)$-uniform-fGA to the $(50 + 50)$-uniform-GA at the target $\phi_s = 89.6$.**

Figure 4.5a indicates the two dynamic GAs on F6 that have different performances. DynamicAUC switches at $\phi_s = 20$ from `(10+5)-two-point-fGA` to `(1+1)EA>0` while dynamicERT switches at $\phi_s = 20$ from `(100+100)-one-point-GA` to `(1+1)EA>0`. We notice that dynamicAUC has a similar convergence process to BSA while dynamicERT does not, which is related to the different $A_1$ parameter settings. Besides, it could be the reason that dynamicERT takes more evaluations to adapt to `(1+1)EA>0`.

Regarding F9, figure 4.5b displays that the efficiency of dynamicAUC overtakes the

other two before reaching the target $f(x) = 98$. The difference is that dynamicERT switch at $\phi_s = 96$ from `(50+1)-uniform-fGA` to `(100+100)-uniform-fGA` while dynamicAUC switch at $\phi_s = 89.6$ from `(50+1)-uniform-fGA` to `(50+50)-uniform-fGA`. The function evaluations of dynamic GAs have a sudden rise after its switch while the BSA doesn't. We also test for dynamicERT switching to `(50+50)-uniform-fGA` at $\phi_s = 96$ which performs worse than before. Under the condition that dynamicERT and dynamicAUC have similar ERT, we consider that `(50+50)-uniform-GA` is not suitable on F9 even it's the best theoretical result from table 4.2.




**(a)** *Fixed-target ERT of GAs on F14* **(b)** *Fixed-target ERT of GAs on F24*

**Figure 4.6: Fixed-target ERTs of GAs on F14 and F24 in dimension $d = 100$. The dynamicAUC in figure 4.6a switches from the $(100+1)\mathbf{EA}_{>0}$ to the $(100+1)$-one-point-GA at the target $\phi_s = 5.1$. The dynamicAUC in figure 4.6b switches from the $(1+1)\mathbf{EA}_{>0}$ to the $(100+1)$-two-point-fGA at the target $\phi_s = 15.7$.**

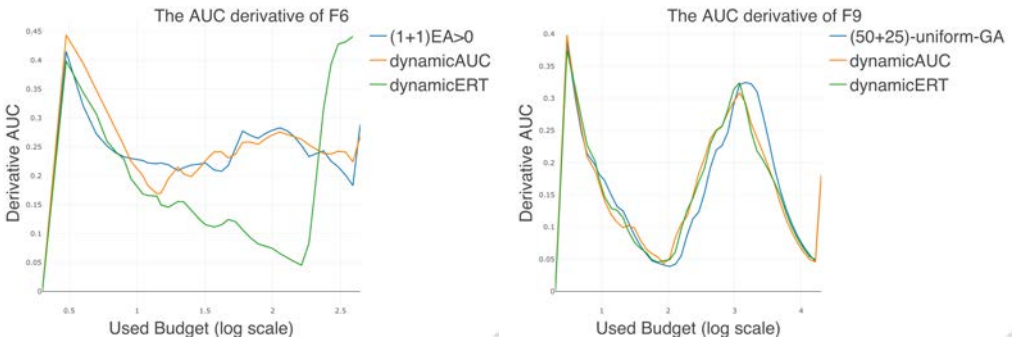
Figure 4.6a presents the ERT curves of two dynamic policies and the BSA of F14. We expect dynamicAUC could perform better than BSA. But unlike F6 or F9, dynamicAUC and dynamicERT perform worse than the BSA for almost the entire process. DynamicAUC switches at $\phi_s = 5.1$ from `(100+1)EA>0` to `(100+1)-one-point-GA`. DynamicERT switches at $\phi_s = 5$ from `(100+50)-one-point-GA` to `(50+25)-uniform-fGA`. F14 presents a W-model transformation to the LeadingOnes (F2) problem, which formulates the step function curve. But as it can be solved early, we need to consider other problems about the impact of population size adaption.

Regarding F24, the worst performance of dynamicAUC, figure 4.6b displays the ERT with an almost vertical ascent after the switch. F24 could not reach the final target due to the vertical rises of dynamicAUC and dynamicERT, and then we set the maximum budget of 500000 as dERT. Before the switch, the two dynamic

GAs performs much better than BSA. DynamicAUC switches at $\phi_s = 15.7$ from `(1+1)EA>0` to `(100+1)-two-point-fGA` while dynamicERT switches at $\phi_s = 15.81$ from `(1+1)fastGA` to `(100+100)-uniform-fGA`. We consider that the extreme cost after the switch is related to the local optima which brings difficulties to convergence [25].

In our scenario, the dynamic policy could combine the excellent efficiency of the two algorithms at different times to obtain a superior dynamic algorithm at the whole process. But F24, the concatenated Trap (CT) problem, is defined by partitioning a length $n$ bit-string into segments of length $k$ and concatenating $m = n/k$ trap functions that take each segment as input. With the complex input, it's hard to make progress with random initialized individuals in the GA solving process. We notice that the point at which the dynamic GAs become particularly inefficient is the switch point. It indicates that the scenario we want is not achieved.

Based on the ERT policy from table 4.1, we noticed that dynamicERT also performs not well as expected. DynamicERT gets ERT value 371 instead of 255 on F6, and 4731 instead 2255 on F9. Besides, though dynamicAUC performs worse than BSA on these two functions, it has a lower ERT value compared with dynamicERT on F6, F9, and F14. Regarding F14 and F24, we notice the same situation: BSA is better than dynamicAUC, which is better than dynamicERT.



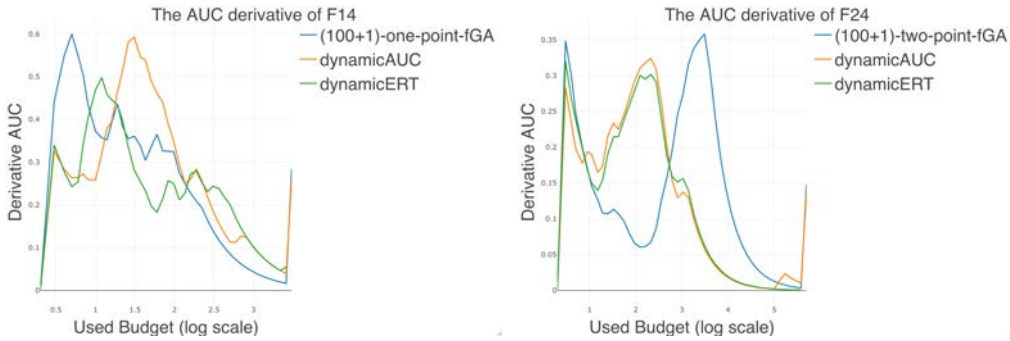

**(a)** *AUC derivative of GAs on F6* **(b)** *AUC derivative of GAs on F9*

**Figure 4.7: AUC derivative of GAS on F6 and F9 in dimension $d = 100$.**

Taking AUC derivative into consideration helps us to find out the reasons about their bad performance. From figure 4.7a we notice that dynamicERT has lower derivative before $10^{2.36} = 229$ evaluations, which is the time that dynamicERT reach $\phi = 26$ in figure 4.5a. After that, dynamicERT presents an efficient period with high deriva-

tive in AUC and low gradient on ERT. Regarding dynamicAUC, it doesn't differ much from BSA and makes no significant progress on ERT. We find that the ERTs corresponding to the switch target $\phi_s$ are 10 and 17.7, which are presented as 1 and 1.24 in AUC derivative curves. From the AUC curves, dynamicAUC has a small increase after switch but dynamicERT has a decrease.

For F9, we notice from figure 4.7b that the three GAs have similar derivative curves and ERT, except BSA performs better from 2.98 to 3.6 of x-axis period with higher derivative and lower ERT. Considering about the switch points of two dynamic GAs, we find that they are about 2.97 and 3.19 of x-axis respectively. The figure indicates that the switch doesn't perform as expected. One reason could be that dynamic GAs transformed to similar parameter settings.




**(a)** *AUC derivative of GAs on F14* **(b)** *AUC derivative of GAs on F24*

**Figure 4.8: AUC derivative of GAS on F25 and F24 in dimension $d = 100$.**

Aiming to know the details about dynamic GAs on F14 and F24, we also plot the AUC derivative figures in figure 4.8. For F14, figure 4.8a presents the interesting curves that BSA performs best before 0.9, then is surpassed by dynamicERT, which is also exceeded by dynamicAUC after 1.2. We take the corresponding switch targets into considerations, finding that the two switch points have less impact on the solving process comparing with other functions above. The reason could be the same as we mentioned about F9.

Figure 4.8b indicates a significant difference between BSA and dynamic GAs. Derivative of BSA is under the other two algorithms before $10^{2.79} = 616$ evaluations. But after the intersection, BSA gets the high derivative which reflects the efficient low gradient in ERT figure 4.6b. In contrast, the AUC derivatives of two dynamic GAs decrease too much, leading to a vertical ascent in ERT. We notice that the ascent in

ERT appears when the function value is about 16, which matches the switching target of dynamicAUC. We consider that it is related to the transformation of policy form $A_1$ to $A_2$ for adapting population size.

**Transformation strategies** As we mentioned in Alg 2, we apply the transformation process for adapting population size from $A_1$ to $A_2$. For instance, dynamic GAs that transform from 1 population size to 100 population size, enlarge by duplicating at first. But the same individuals in a population would be meaningless for crossover operator. Besides, the loss of cumulative mutation due to transformation takes more time to solve the problems. In this way, we try different strategies of switching transformation.

- Half by copy, half by initialization
- Half by copy, half by mutation randomly
- Half by copy, half by mutation once

It is different between mutation randomly and mutation once. The number of mutated individuals depends on the mutation strength if we do mutation randomly, while all the rest individuals would be mutated once if we use the latter strategy. We run those experiments with all the strategies above. Regarding half of initialization, we notice that it destroy the efforts from previous operations, including cumulative mutation and some competitive individuals. We gain slight improvement with half by mutation randomly because the mutation strength L we set is small for the rest half of the individuals.

We apply the the third strategy in our experiment and the presented data are based on it. This strategy provide a baseline by duplicating and a potential improvement with the rest half of individuals. But it is obvious that the unexpected performance on F24 and other functions that need enlargement transformation hasn't been solved. Future work can concentrate on the way for adapting population size transformation.

**Summary** We tried different transformation strategies to improve the performance of dynamic GAs on different functions, but the results are not ideal. The theoretically good results of the two types of dynamic policies are under-performed in practice due to the population size expansion. Despite this challenge, we detect that even under this situation, dynamicAUC policy performs better than dynamicERT.

# Chapter 5

# Conclusions and Future Work

The purpose of the current project is to explore how AUC could be an effective performance measure in helping resolve dynAS problems. The second aim of this project is to investigate the effects of different switch targets which are obtained by ERT and AUC respectively. We calculate the theoretical AUC results on 25 different black-box optimization problems with different policies in order to make comparisons with empirical results.

Firstly, the investigation of AUC shows that it is an effective performance measure for dynAS problems in many problems, i.e. F1, F2, F4, F5, etc. Comparing with ERT, the switch target $\phi_s^{(\mathrm{AUC})}$ which obtained via AUC value is smaller than $\phi_s^{(\mathrm{ERT})}$. With an earlier switch target, the performance of AUC selection policy surpassed those detected from ERT. We consider that ERT can be obtained after a whole run while AUC can be calculated after each evaluation. The lag in the ERT calculation causes it to have the same lag as a performance measure, whereas the AUC does not have this drawback. But AUC needs more settings including budget set and target set. If the budget sets of two involved algorithms are different, the AUC selection policies would not be applied to the algorithms. And if the target set is not suitable, for instance, too small, the value of AUC could be too high to distinguish policies.

On the one hand, we obtain good responses from AUC selection policies. On the other hand, the results indicate that a proper population transformation strategy is necessary for the dynAS problems since we observe the worse performance of the policies that switch between different population sizes compared to BSA. In detail, the transformation from a small population to a large population leads to that. We try different strategies, from copying and mixing with mutations, but the two strategies show

similar performance and do not the solve the problem. Similar issues were detected from the study of ERT selection policy which we refer to. Despite its limitations, the study certainly adds to our understanding of the dynAS problems via AUC.

The previous unexpected results on F24 and F25 indicate that the best policy in theory is not reliable in some problem. A bigger set of algorithms $\mathcal{A}$ may help. In fact, we add constraints on the set of algorithms while we choose to use one-shot GA with preset parameters. It hides the potentially optimal policies as they may not use the preset parameters.

Our results revealed that the static AUC results might lead to a lousy policy due to the small AUC range of values. The limitation of the target set may cause a one-side solution such as F24, in which we could not find a better dynamic policy than BSA.

We should also consider the self-adaptive parameter selection based on AUC, as the derivative of AUC provides the trend of GAs. AUC seems to be a good measure reflecting convergence that can be used in machine learning. Reinforcement learning (RL) can be a metric for algorithm selection problems because the self-adaptive process can be regarded as a simple process of RL with an inner reward fitness function. Future work can focus on constructing a reward function to distinguish different policies as known as agents in RL theory.

Besides, the issue of strategies for selecting parameters of $\pi$ is intriguing and could be usefully explored in further research. And we can focus on better strategies for adapting population size to resolve the problem caused by too much variation in population size. Moreover, customising fitness functions for problems suffering from local optima could be combined with the transformation issue. As we are using the basic fitness function of 25 PBO problems, well constructed fitness function may benefit the process of solving problems and avoid local minima.

# Bibliography


[1] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[2] Prasanna Balaprakash, Mauro Birattari, and Thomas Stützle. Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. In *International workshop on hybrid metaheuristics*, pages 108–122. Springer, 2007.

[3] Nils Aall Baricelli. Numerical testing of evolution theories, part ii preliminary tests of performance. *Symbiogenesis and terrestrial life, Acta Biotheoretica*, 16:99–126, 1962.

[4] Nils Aall Barricelli. *Symbiogenetic evolution processes realized by artificial methods*. 1957.

[5] William Jay Conover. *Practical nonparametric statistics*, volume 350. john wiley & sons, 1999.

[6] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. Fast genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 777–784, 2017.

[7] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics. *arXiv e-prints:1810.05281*, October 2018.

[8] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, Ofer M. Shir, and Thomas Bäck. Benchmarking discrete optimization heuristics with iohprofiler. *Appl. Soft Comput.*, 88:106027, 2020.

[9] Alex Fraser, Donald Burnell, et al. Computer models in genetics. *Computer models in genetics.*, 1970.

[10] Alex S Fraser. Simulation of genetic systems by automatic digital computers i. introduction. *Australian journal of biological sciences*, 10(4):484–491, 1957.

[11] David J Hand and Robert J Till. A simple generalisation of the area under the roc curve for multiple class classification problems. *Machine learning*, 45(2):171–186, 2001.

[12] William E Hart and John M DeLaurentis. Convergence of a discretized self-adapative evolutionary algorithm on multi-dimensional problems. Technical report, Sandia National Laboratories (SNL), Albuquerque, NM, and Livermore, CA ..., 2003.

[13] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[14] Holger H Hoos. Automated algorithm configuration and parameter tuning. In *Autonomous search*, pages 37–71. Springer, 2011.

[15] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

[16] Thomas Jansen and Christine Zarges. Analysis of evolutionary algorithms: From computational complexity analysis to algorithm engineering. In *Proceedings of the 11th workshop proceedings on Foundations of genetic algorithms*, pages 1–14, 2011.

[17] Donald E Knuth. Computer programming as an art. In *ACM Turing award lectures*, page 1974. 2007.

[18] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[19] Silja Meyer-Nieberg and Hans-Georg Beyer. Self-adaptation in evolutionary algorithms. In *Parameter setting in evolutionary algorithms*, pages 47–75. Springer, 2007.

[20] Jonas Močkus. On bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*, pages 400–404. Springer, 1975.

[21] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[22] Chuan-Kang Ting. On the mean convergence time of multi-parent genetic algorithms without selection. In *European Conference on Artificial Life*, pages 403–412. Springer, 2005.

[23] Diederick Vermetten, Hao Wang, Thomas Bäck, and Carola Doerr. Towards dynamic algorithm selection for numerical black-box optimization: investigating bbob as a use case. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 654–662, 2020.

[24] Wikipedia contributors. Genetic algorithm — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Genetic_algorithm&oldid=1115108091, 2022. [Online; accessed 16-November-2022].

[25] Furong Ye, Carola Doerr, and Thomas Bäck. Leveraging benchmarking data for informed one-shot dynamic algorithm selection. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 245–246, 2021.