國立臺灣大學電機資訊學院電機工程學系
碩士論文
Department of Electrical Engineering
College of Electrical Engineering and Computer Science
National Taiwan University
Master Thesis

基於子空間映射與多臂吃角子老虎機技術之實數最佳化

Real-valued Optimization by Subspace Projection and Multi-armed Bandit Techniques

彭俊人
Chun-Jen Peng

指導教授：于天立博士
Advisor: Tian-Li Yu, Ph.D.

中華民國 106 年 7 月
July, 2017

摘要

本論文提出了一新的實數多模態 (multimodal) 最佳化方法。大多數的現實問題都可以由多個單峰問題合成。而對於實數最佳化，我們較感興趣的是多模態且可被拆解成多個階層化的單峰問題。然而，一旦面對多模態問題，我們就必須面對利用性 (exploitation) 與探索性 (exploration) 的問題。本論文提出一解決多模態問題的技巧。此技巧結合了階層式聚合 (hierarchically clustering) 與最小描述長度 (minimum description length)，來辨識搜尋空間中可能的單峰模型。接著，我們使用一加一演化策略 ((1+1)-Evolutionary Strategy) 來最佳化一齊次座標 (homogeneous coordinate) 上的線性轉換矩陣，並利用此矩陣將原搜尋空間投影到另一有良好邊界定義的子空間。此投影嘗試定義出只包含一單峰之感興趣區域 (region of interest)。最後，我們提出一新的多臂吃角子老虎 (multi-armed bandit) 技術，來分配給各個子空間的資源，以最大化獲得全域最佳解的機率，我們將此結果與自適應共變異數矩陣演化策略 (Covariance Matrix Adaptation Evolution Strategies, CMA-ES)、標準粒子群演算法 (Standard Particle Swarm Optimization 2011)、與實數域蟻群演算法 (Ant Colony Optimization for Continuous Domain, ACO_R) 結合，並利用 CEC 2005 實數最佳化測試問題來比較結果。

Abstract

This thesis presents a new technique for real-valued multimodal optimization. Most of the real-world problems can be described as a composition of unimodal subproblems. For real-valued optimization, we are interested in problems that are composed of *multiple hierarchically decomposable* unimodals. However, allocating resources to exploit different unimodals leads to the common dilemma between *exploration* and *exploitation*. This thesis proposed some new techniques that aim to solve multimodal problems more efficiently. A new technique combining hierarchical clustering and minimum description length (MDL) is proposed to help identify potential unimodals in the search space. Then, a linear transform matrix in homogeneous coordinate, optimized by the (1+1)-Evolutionary Strategy, projects the search space to a subspace with well-defined boundary. This projection aims to define a region of interest (ROI) that isolates a unimodal subproblem. Finally, a new multi-armed bandit technique, aiming to maximize the probability of acquiring the global optimum, is proposed to allocate resources for each unimodal. We combined our new techniques with Covariance Matrix Adaptation Evolution Strategy (CMA-ES), Standard Particle Swarm Optimization (SPSO) 2011, and Ant Colony Optimization for Continuous Domain (ACO_R). The results are evaluated with the CEC2005 Special Session on Real-Parameter Optimization benchmark problems.

Contents

摘要 iii

Abstract v

1 Introduction 1

1.1 Thesis Objectives 2

1.2 Roadmap 3

2 Real-valued Optimization Algorithms 5

2.1 Covariance Matrix Adaptation Evolution Strategy 5

2.1.1 (1+1)-ES 6

2.1.2 CMA-ES 7

2.2 Standard Particle Swarm Optimization 9

2.2.1 Initialization of the swarm 9

2.2.2 Velocity update equations 11

2.2.3 The adaptive random topology 13

2.3 Ant Colony Optimization for Continuous Domain 14

2.3.1 Discrete solution construction and pheromone update 15

2.3.2 Continuous solution construction and phermone update 16

3 Clustering Techniques 19

3.1 K-Means Clustering 20

3.2 Heirarchical Clustering 22

3.3 Minimum Description Length 24

3.3.1 Data description accuracy 24
3.3.2 Model description efficiency 26
3.3.3 Determining model with MDL 27

4 Linear Projection **29**
4.1 Affine Transformation 30
4.2 Projective Transformation 33
4.3 Optimization for Projection Matrix 35
4.3.1 Loss function 35
4.3.2 Optimization algorithms 38

5 Multi-armed Bandit Algorithms **39**
5.1 The Multi-armed Bandit Problem 40
5.2 Bandit Algorithms Overview 41
5.3 The New Bandit Technique 42

6 The New Multimodal Optimization Technique **45**
6.1 Framework of the New Multimodal Optimization Technique 45
6.2 Initialization and Unimodal Identification 47
6.3 Define Region of Interest and Construct Arms 47
6.4 Remain Evaluations Allocation and Recluster 48

7 Experiments **51**
7.1 CEC2005 25 Benchmark Problems 51
7.2 Experiment Settings 52
7.3 Experiment Results 53

8 Conclusion **63**

Bibliography **65**

List of Figures

2.1 The one fifth rule of (1+1)-ES. 6

2.2 Three major elements of CMA-ES. 8

2.3 Velocity update of SPSO 2011. 11

2.4 Different topologies used by PSO. 13

2.5 Probability of different numbers of informants in the random topology. . . 14

2.6 Comparison of discrete PDF continuous Gaussian kernel PDF. 15

2.7 The solution archieve of ACO_R 17

3.1 Comparing K-Means clustering with weighted Gaussian Distribution on CEC2005 F1 Problem 21

3.2 Comparing K-Means clustering with weighted Gaussian Distribution on CEC2005 F11 Problem 22

3.3 Before and after applying Minimal Description Length on CEC2005 F19 Problem 23

3.4 Before and after applying Minimal Description Length on CEC2005 F1 Problem 24

4.1 Projecting inseparable problems onto subspace. 30

4.2 Some common affine transformation 31

4.3 General affine transformation in 2D 32

4.4 Affine transformation and projective transformation comparison. 33

4.5 Projective transformation matrix. 34

4.6 Comparison between inverse affine transform and inverse projective transformation. 34

4.7 Snapshot of optimization for the projection matrix of the red ROI. 37

5.1 Probability of regret. 43
5.2 Probability of regret for all allocation combinations. 44

7.1 Average error of Problem 1 to Problem 8. 55
7.2 Average error of Problem 9 to Problem 16. 56
7.3 Average error of Problem 17 to Problem 24. 57
7.4 Average error of Problem 25. 58
7.5 Median error of Problem 1 to Problem 8. 59
7.6 Median error of Problem 9 to Problem 16. 60
7.7 Median error of Problem 17 to Problem 24. 61
7.8 Median error of Problem 25. 62

List of Tables

7.1 Summary of the parameters used by ACO_R 53

Chapter 1

Introduction

Multimodal problems that contain more than one region or more than one local optimum are common in the real world. *Rastrigin function*, a hierachical problem which is composed of a larger unimodal and multiple smaller unimodals are also well-known in real-valued optimization. In order to tackle these multimodal problems with a limited amount of resources, one needs to decide how much to invest for searching a better unimodal, while still maintain enough of exploitation on each potential unimodal.

The difficult part is that searching for a new unimodal and exploiting the best unimodal often require opposite searching behaviors. For a fixed amount of total evaluations, spending more time searching on one unimodal implies less attention on exploring other possible unimodals in the search space. This is the common dilemma between *exploration* and *exploitation* for real-valued optimization algorithms. Also, during different phases of optimization, the ratio between exploring new unimodals and exploiting the current best unimodal should be altered. Exploration is more important in the beginning, while exploitation is more required in the end. What makes this problem even harder is that in order to decrease the number of total evaluations, the size of the population are often limited in the real-valued optimization algorithms. This means that as the number of unimodals to invest increases, the number of particles on each unimodal decreases. This weakens the exploitation ability on each unimodal, making it less likely to identify new potential regions on each unimodal.

We are more interested in hierarchically decomposable problems. With a certain amount

of hints, we should be able to focus on one of the many potential unimodals, and identify when to invest more on certain unimodals to iteratively discover more smaller unimodals. Also, we would like to only focus on some regions instead of the whole search spaces, in order to gain performance. This requires detection of promising areas, and search space splitting techniques. Moreover, after identifying certain interesting regions, we also need to decide how to allocate our resources. Given the abilities described above, we can pay more attention on the promising regions and iteratively break down a difficult multimodal problem into smaller and easier unimodal problems.

1.1 Thesis Objectives

We propose some techniques to break down multimodal problems into several unimodal problems in smaller non-overlapping subspaces. We believe that solving one unimodal problem on a smaller subspace is easier than tackling the complete search space as a whole. This technique can **identify potential unimodal**, **define a region of interest** to exploit, and **allocate resources** according to remaining evaluations.

Identifying a *unimodal* depends on the sampling frequency and the underlying model. Clustering is a common way for discovering potential unimodals. We adopt the hierarchical clustering technique to discover unimodals in the fitness landscape. We also use weighted normal distribution as the underlying consumption for unimodal, and reject overfitting models with minimum description length.

After identifying the unimodals by clustering, we can split the search space into different regions of interest with linear projection. We use different projection matrices to project the original search space onto several subspaces with well-defined boundaries. We also optimize the projection matrix so that each region would have minimum overlapping with others. Searching in the smaller subspace enhances the probability of finding optimum. The well-defined boundaries are also needed for some algorithms. The projection might also rotate or shear the original search space, making some inseparable problems easier to solve.

With multiple subspace to search, we propose a new Multi-armed Bandit (MAB) tech-

nique that optimize the resource allocation to get the greatest probability of obtaining the global optimum. Multi-armed Bandit algorithms are suitable for this scenario, since it learns models from outcomes while the actions do not change the state of the world. When there are still plenty of evaluations left, we should prefer exploration over exploitation and search equally on all subspaces. However, when few evaluations remain, we should focus on the current best unimodal and try to exploit it to get the best fitness with the remaining evaluations.

1.2 Roadmap

This thesis is composed of eight chapters.

Chapter 2 presents three optimization algorithms that are used for comparisons. We introduce the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES), the Standard Particle Swarm Optimization (SPSO), and the Ant Colony Optimization for Continous Domain (ACO_R). These three algorithms each have different characteristics due to different underlying models. We get a glimpse of how some common real-valued algorithms tackle multimodal problems.

Chapter 3 first discuss our experience on using K-Means to identify unimodals. Later, a more successful attempt of using the hierarchical clustering is presented. Finally, we give details about Minimal Description Length and how it reduces number of clusters.

Chapter 4 first describes four basic affine transformation: translation, rotation, scaling and shearing. Then the projective transformation and homogeneous coordinate are presented. Finally, we give details about how we optimize our projection matrix in order to obtain ROIs with minimal-overlapping.

Chapter 5 first describes the Multi-armed Bandit Algorithm, A brief introduction of some common MAB algorithms are also given. Then we present our new bandit techniques, that focus on acquiring the best rank in the cluster.

Chapter 6 gives details of our new technique. It first describes the framework of the new technique. Then a detailed process of initialization and unimodal detection is given. The implementation detail of how to optimize the ROI and how to initialize arms are given.

Later, we discuss about how to decide which arm to pull in order to match the time-variant remain resource allocation.

Chapter 7 introduces the CEC 2005 benchmark problems that we used for experiment. Then we give details about the experiment setup for different algorithms and the termination criteria. The results are shown in figures that indicates the error after the same amount of evaluations.

Chapter 8 summarizes this thesis. The conclusion and contributions are also given. Some further improvements and future works are also discussed at the end.

Chapter 2

Real-valued Optimization Algorithms

A real-valued optimization algorithm minimize a given *objective function* in the continuous domain. It searches the continuous domain by iteratively asking for the fitness of solutions. We often evaluate an algorithm by the *number of function evaluations* it consumes to obtain the global optimum solution. This task is often described as the *black box optimization*, since the algorithm has no prior knowledge about the problem characteristics or structures. That means the gradient information might not be useful or even not available, so hill-climbing algorithms do not work. The problem may be non-convex, multimodal, noisy and discontinuous. In order to solve these kinds of problems, different algorithms have different assumptions about the problem structure. They also use different stochastic models and update methods to try to converge to the global optimum as fast as possible. These algorithms are often used for parameter and model calibration. The following section describes three kinds of milestone evolutionary algorithms that are commonly used as the baseline for real-valued optimization.

2.1 Covariance Matrix Adaptation Evolution Strategy

The *Evolutionary Strategies* (ES) samples new search points with a normal distribution. Each d-dimensional sample x_i in a population with size λ is drawn from a multivariate

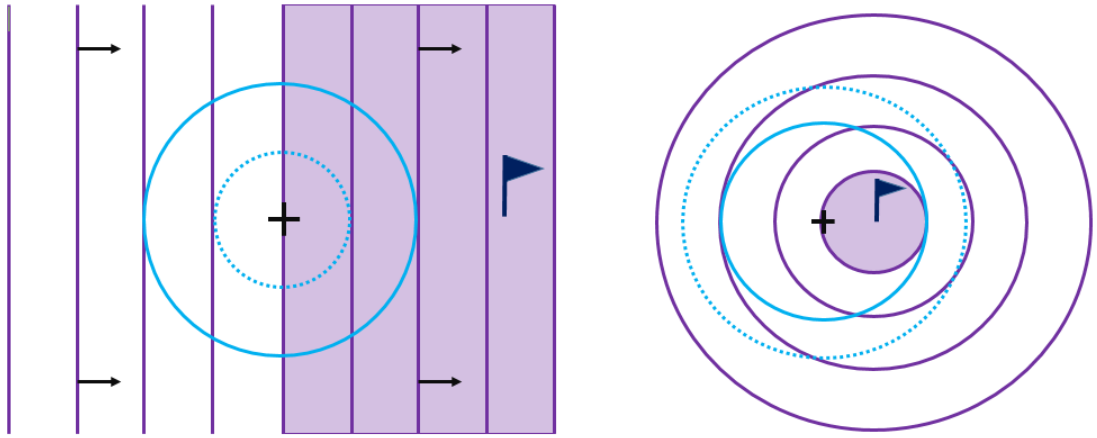


Figure 2.1: The one fifth rule of (1+1)-ES.

normal distribution as following:

$$x_i \sim m + \sigma N_i(0, C),$$

where $m \in \mathbb{R}^d$ is the mean, $\sigma \in \mathbb{R}_+$ is the step-size, and $C \in \mathbb{R}^{d \times d}$ is the covariance matrix. The mean vector represents the positions where the best solution is most likely to appear. The step-size determines the area to search and modifies exploration and exploitation behaviors. The covaraince matrix determines the shape of the distribution ellipsoid.

2.1.1 (1+1)-ES

One of the simplest Evolutionary Strategy is **(1+1)-ES**. It samples one offspring x from parent m according to the equation described above. If the offspring x is better than m, it replaces m to be the new mean of the next sampling distribution. It also follows the *one-fifth success rule* as illustrated in Figure 2.1. The one-fifth success rule increases the step-size by 1.5 if the current sample is not better than the previous best sample, *i.e.* the mean m. We believe the best solution, denoted as the flag in Figure 2.1(a), should be somewhere outside of our current distribution and we should increase step-size to explore more search space. The one-fifth success rule decreases the step-size by $1.5^{-1/4}$ if the current sample is better than the previous best sample. This means that we believe the best solution,

illustrated as the flag in Figure 2.1(b), should be within the sampling distribution, and we should decrease the step-size to further focus on the current region. Here we describe the (1+1)-ES with one-fifth success rule with independent restarts [1]. The pseudo code of (1+1)-ES is given in Algorithm 1.

Algorithm 1: (1+1)-ES with 1/5 success-rule

$\boldsymbol{X}_n$: solution of the n^{th} iteration, σ_n: step size of the n^{th} iteration,
$N(\mathbf{0}, \boldsymbol{I})$: multivariant normal distribution with mean vector $\mathbf{0}$
and identical covariance matrix $\boldsymbol{I}$.

input : f: evaluation function
output: X_{n+1}: best solution

Initialize $\boldsymbol{X}_0, \sigma_0$
while *termination criterion not met* **do**
  $\widetilde{\boldsymbol{X}}_n = \boldsymbol{X}_n + \sigma_n N(\mathbf{0}, \boldsymbol{I})$
  if $f(\widetilde{\boldsymbol{X}}_n) \leq f(\boldsymbol{X}_n)$ **then**
    $\boldsymbol{X}_{n+1} = \widetilde{\boldsymbol{X}}_n$
    $\sigma_{n+1} = 1.5\sigma_n$
  else
    $\boldsymbol{X}_{n+1} = \boldsymbol{X}_n$
    $\sigma_{n+1} = 1.5^{-1/4}\sigma_n$
return $\boldsymbol{X}_{n+1}$

2.1.2 CMA-ES

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is an Evolutionary Strategy proposed by Hansen in 2003 [7]. It adopts a multivariate normal distribution to generate *isotropic* search points which do not favor any direction. The multivariate normal distribution is widely observed in the nature and relatively easy to design and analysis for algorithms. CMA-ES modifies three parameters to control the multivariate normal distribution: the mean $m \in \mathbb{R}^d$, the step-size $\sigma \in \mathbb{R}_+$, and the covariance matrix $C \in \mathbb{R}^{d \times d}$.

The *mean* indicates the translation of the normal distribution model. It indicates the most favorable position that should be sampled with the largest density. It is updated with the weighted mean of all the positions according to their fitness. The *step-size* controls the state of exploration or exploitation. An ideal step-size control gradually decreases the searching region for exploitation. One of the easiest step-size control method is the one-

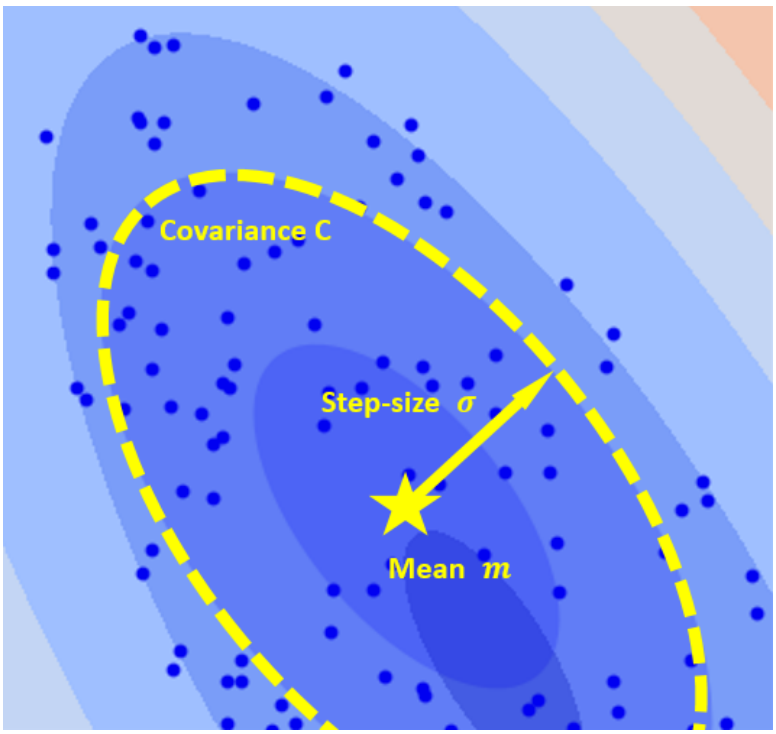


Figure 2.2: Three major elements of CMA-ES.

fifth success rule, mentioned above. Later a cumulative step-size adaptation is proposed in Cumulative Step-size Adaptation Evolutionary Strategy (CSA-ES) [8], a predecessor of CMA-ES. The cumulative step-size adaptation measures the pathway of the mean vector in the generation sequence. Adapting the step-size according to the *evolution path* enables translating more efficiently to a desired location. Later, it also allows perpendicular around the desired region. The *covariance matrix* indicates the shape of the normal distribution model. Adapting the covariance matrix allows us to search in a region that is more coherent to the fitness landscape, This adaptation follows a natural gradient towards the direction of optimum solution. It can also learn a rotated representation of the problem and be coordinate independent. By adjusting these three parameters, shown in Figure 2.2, CMA-ES is extremely powerful on solving unimodal problems.

CMA-ES adopts a small *population size* to reduce function evaluations and increase the frequency of updating the underlying models. The default population size λ for a D-dimension problem, described in [6], is

$$\lambda = 4 + \lfloor 3 \log(D) \rfloor.$$

However, recent research suggest that it is benefitial to gradually increase the population size after random restart [2]. This parameter-less adaptation enables few evaluations consumption for easy problems, yet is also able to solve more difficult problems with a larger population size later.

2.2 Standard Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a swarm intelligence optimization algorithm. It was first proposed by J. Kennedy and R. C. Eberhart in 1995 [11] to simulate the foraging behavior of bird flocks. The swarm is composed of *particles* that move around in a given multi-dimensional *search space* to find the best solution. Each particle updates its velocity according to its historical experience, as well as the information of the neighboring particles. The neighborhood of a particle is a set of information links defined by the *swarm topology*. PSO iteratively updates the swarm topology, the velocities, and the positions of each particle until the global optimum solution is found.

Throughout the years, numerous variants of PSO have been proposed to improve performance. As a result, a *standard* PSO, composed of clear principals, is needed as the baseline for comparison. Standard PSO (SPSO) provides a well defined version that follows the common principals of PSO design. It is intended to be a milestone with simple and clear implementation for future comparison. So far, there have been three successive versions of standard PSO: SPSO 2006, SPSO 2007, and SPSO 2011. The underlying principals of these three algorithms are generally the same as all PSO variants. The exact formula and implementation are slightly different due to latest theoretical progress. A detailed description of SPSO 2011 [23] is given in the following paragraphs.

2.2.1 Initialization of the swarm

For a search space E with a given dimension D, E is confined by a set of minimum and maximum bounds in each dimension. The hyperparallelepid search space E can be

formally defined as the Euclidean product of D real intervals [4]:

$$E = \bigotimes_{d=1}^{D} [min_d, max_d].$$

For each position x within the multi-dimensional search space E, there exists a corresponding numerical value $f(x)$, *i.e. fitness*. A swarm is composed of particles, which explore different positions in the search space to find the best corresponding fitness value. At time t, each particle in the swarm possesses the following vectors with D coordinate:

- $x_i(t)$ is the **position** of the particle i at time t.
- $v_i(t)$ is the **velocity** of the particle i at time t.
- $p_i(t)$ is the **previous best position** the particle i had been to, at time t.
- $l_i(t)$ is best position of all the previous best positions in the **neighborhood** of particle i at time t.

Let $U(min_d, max_d)$ be a random number drawn from a uniform distribution within $[min_d, max_d]$, and $N_i(t)$ be a set of neighbors of particle i at time t defined by the swarm topology. In SPSO 2011 [4], each particle is initialized with a random position and velocity defined as following:

$$\begin{aligned}
x_i(0) &= U(min_d, max_d), \\
v_i(0) &= U(min_d - x_{i,d}(0), max_d - x_{i,d}(0)), \\
p_i(0) &= x_i(0), \\
l_i(0) &= argmin_{j \in N_i(0)}(f(p_j(0))).
\end{aligned}$$

The swarm size of SPSO 2011, denoted as S, is suggested as

$$S = 40,$$

yet it can also be defined by user [4].

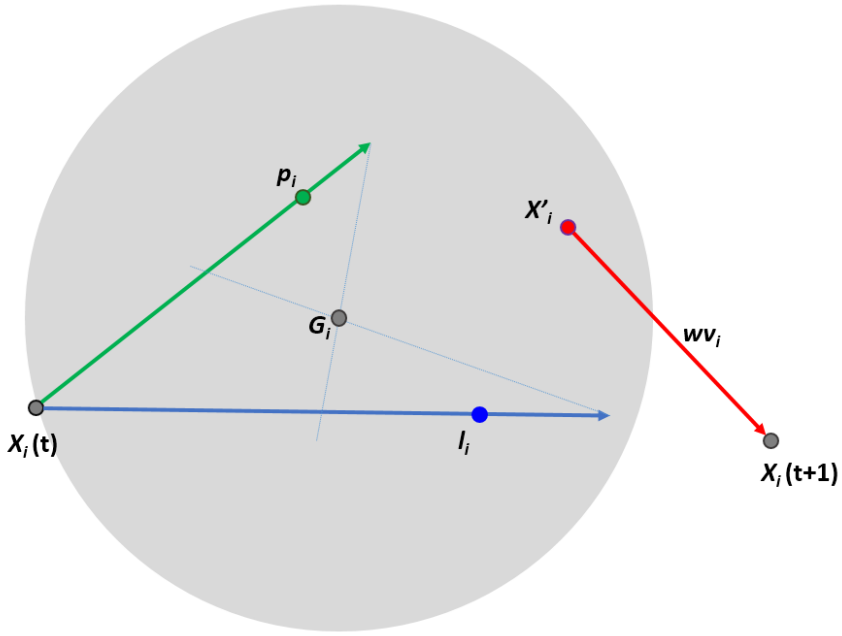


Figure 2.3: Velocity update of SPSO 2011.

2.2.2 Velocity update equations

Velocity update equations differs in different variations of PSO, since it is the core procedure of optimization that determines the performance of PSO. To prevent premature convergence, SPSO follows a random permutation order to update the positions of particles in the swarm [4]. The position of particle i at time $t+1$ depends on the previous position and the current velocity:

$$x_i(t+1) = x_i(t) + v_i(t+1).$$

The velocity of particle i at time $t+1$ can be described as a combination of three vectors:

$$v_i(t+1) = wv_i(t) + \alpha(p_i(t) - x_i(t)) + \beta(l_i(t) - x_i(t)),$$

where w mimics the momentum of the particle, and α, β describes how successive actions are effected by the personal previous best position and the neighborhood previous best position.

In SPSO 2011, the velocity update equations eliminates the coordinate dependency by

creating a hypersphere according to x_i, p_i and l_i, shown in Figure 2.3. The hypersphere is defined as:

$$H_i(G_i, ||G_i - x_i||),$$

with center G_i and radius $||G_i - x_i||$. When the personal best $p_i(t)$ is not the neighborhood previous best $l_i(t)$, the center G_i is defined as:

$$G_i = \frac{1}{3}(x_i + (x_i + c(p_i - x_i)) + (x_i + c(l_i - x_i))) = x_i + \frac{c}{3}(p_i + l_i - 2x_i).$$

If the personal best is the best in the neighborhood, *i.e.* $p_i(t) = l_i(t)$, the center G_i is defined as:

$$G_i = \frac{1}{2}(x_i + (x_i + c(p_i - x_i))) = x_i + \frac{c}{2}(p_i - x_i).$$

In both cases, the acceleration constant c is:

$$c = \frac{1}{2} + \ln(2) \simeq 1.193.$$

As shown in Figure 2.3, a random sample $x_i^{'}$ is drawn from the hypershpere with uniform random direction and uniform radius:

$$r = U(0, ||G_i - x_i||).$$

Therefore, the velocity update equation and new position of SPSO 2011 is defined as:

$$v_i(t+1) = wv_i(t) + x_i^{'}(t) - x_i(t),$$
$$x_i(t+1) = x_i(t) + v_i(t+1) = wv_i(t) + x_i^{'}(t),$$

where the inertia weight is also:

$$w = \frac{1}{2\ln(2)} \simeq 0.721.$$

As mentioned before, the search space is defined as a hyperparallelepid with confine-

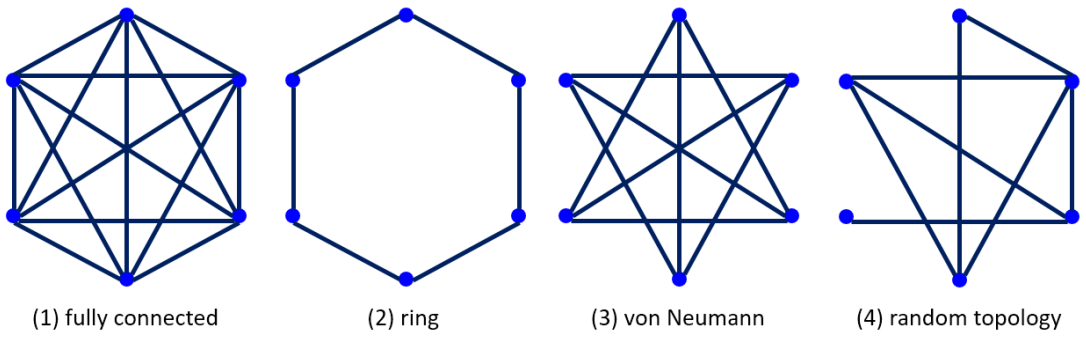


Figure 2.4: Different topologies used by PSO.

ment $[min_d, max_d]$ in each dimension d. Therefore, after calculating the new velocities and positions of particles, we have to go through confinement check to make sure all particles stay inside the search space. There are multiple options to handle confinement in SPSO 2011. The following method described in [4] treat boundary as "wall", so that all particles bounce back with half of previous velocity after they reach the border. For the new position $x_{i,d}(t+1)$ of particle i in dimension d,

$$
\begin{aligned}
if(x_{i,d}(t+1) < min_d), x_{i,d}(t+1) &= min_d, \\
v_{i,d}(t+1) &= -0.5 v_{i,d}(t+1). \\
if(x_{i,d}(t+1) > min_d), x_{i,d}(t+1) &= max_d, \\
v_{i,d}(t+1) &= -0.5 v_{i,d}(t+1).
\end{aligned}
$$

2.2.3 The adaptive random topology

The swarm topology defines "who informs whom" to help distribute the information of optimum solution in the swarm. Multiple topologies shown in Figure 2.4 are adopted in different version of PSO. The random topology adopted in SPSO 2011 is updated under two circumstances [4]:

- at the very beginning
- after an iteration with no improvement of the best known fitness value

For each particle p_i in a swarm of n particles $S = \{p_1, p_2, ..., p_n\}$, a good topology

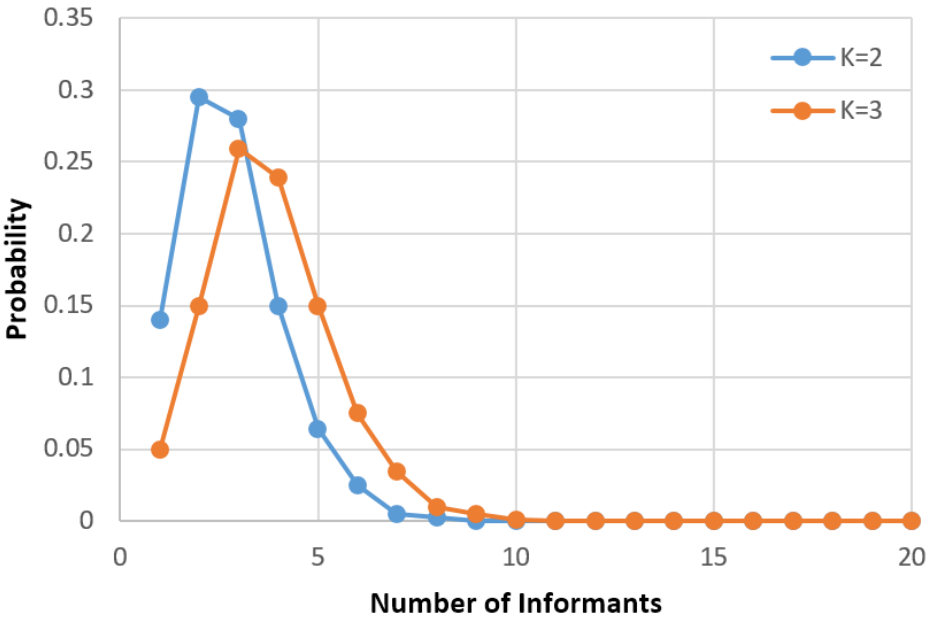


Figure 2.5: Probability of different numbers of informants in the random topology.

should allow each p_i to inform itself, along with a set of randomly selected particles from the other $n-1$ particles [3]. We would like most of the particles to be informed by two to four particles in the topology. Some of the particles can have no informant except for itself. There should also exist a non-zero probability for a particle to be informed by all the other particles.

In order to construct a random topology that satisfy the desirable features mentioned above, Clerc proposed a method [3] that results in a distribution of the sum of $n-1$ independent binary Bernoulli variables, shown in Figure 2.5. We use the distribution $K = 3$, meaning that each particle tells three other particles about their previous best fitness and position.

2.3 Ant Colony Optimization for Continuous Domain

Ant Colony optimization (ACO) is first proposed by Dorigo [5] to solve combinatorial optimization problems, including scheduling, routing, and timetabling. These problems aim to find optimal *combinations* or *permutations* of finite sets of available components. Inspired by the foraging behavior of natural ants, ACO mimics the pheromone deposition

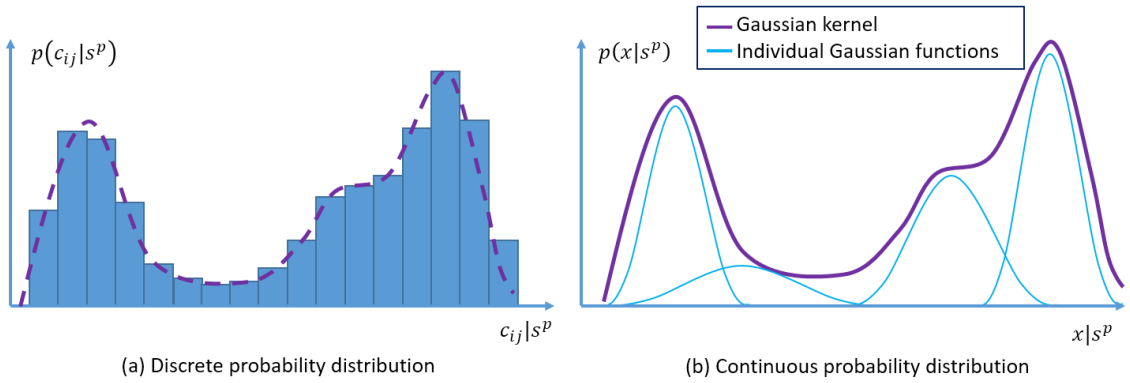


Figure 2.6: Comparison of discrete PDF continuous Gaussian kernel PDF.

of ants along the trail to a food source. The deposited pheromone, which indicates the quantity and quality of the food, creates an indirect communication among ants and enables them to find the shortest paths. Two major procedures: *solution construction* and *pheromone update*, are detailed in the following paragraph.

2.3.1 Discrete solution construction and pheromone update

For *solution construction*, we first consider a search space $\boldsymbol{S}$ defined over a finite set of all possible *solution components*, denoted by $\boldsymbol{C}$. Each solution component, denoted by c_{ij}, is a decision variable X_i instantiated with value $v_i^j \in \boldsymbol{D}_i = \{v_i^1, ..., v_i^{|\boldsymbol{D}_i|}\}$. To construct a new solution, an artificial ants starts with an empty partial solution $s^p = \emptyset$. During each construction step, the partial solution s^p is extended with a feasible solution from the set $N(s^p) \in \boldsymbol{C} \setminus s^p$. The probabilistic pheromone model adopted for selecting a feasible solution from $N(s^p)$ can be defined as follows:

$$p(c_{ij}|s^p) = \frac{\tau_{ij}^{\alpha} \cdot \eta(c_{ij})^{\beta}}{\sum_{c_{i\ell} \in N(s^p)} \tau_{i\ell}^{\alpha} \cdot \eta(c_{i\ell})^{\beta}}, \forall c_{ij} \in N(s^p),$$

where τ_{ij} is the pheromone value associated with component c_{ij}, and $\eta(\cdot)$ is a weighting function. α and β are positive parameters which determine the relation between pheromone and heuristic information.

The *pheromone update*, also known as pheromone evaporation, acts as solution selection. It increases the pheromone values associated with promising solutions and de-

creases the pheromone values associated to the bad ones. This is achieved by increasing the pheromone levels of more preferred solutions by $\Delta\tau$. Let $\rho \in (0, 1]$ be the *evaporation rate*. If the pheromone value is associated with a chosen good solution s_{ch}, the pheromone value is updated as following:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho\Delta\tau,$$

while the other unrelated pheromone values are updated as:

$$\tau_{ij} = (1 - \rho)\tau_{ij}.$$

Thus, the discrete pheromone model shown in Figure 2.6(a) can be updated to emphasize the good solutions.

2.3.2 Continuous solution construction and phermone update

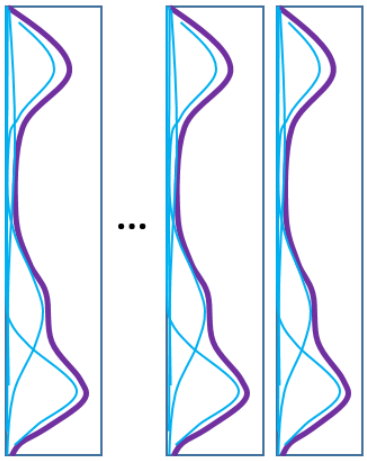
Over the years, multiple approaches of extending the ACO on continuous domain have been given. One of the most successful version is ACO$_R$, proposed by Socha and Dorigo in 2008 [18]. It extends ACO to the continuous domain without making any major conceptual change to its structure. The fundamental idea underlying ACO$_R$ is to substitute the discrete probability distribution with a continuous PDF in each dimension, demonstrated in Figure 2.6. ACO$_R$ keeps a solution archive of **top-k solutions** it has seen so far, shown in Figure 2.7. All the solutions are sorted(ties are broken randomly) according to their fitness $f(s)$, so that the best solution stays at the top of the archive. For the i-th dimension, a Gaussian kernel PDF $G^i(x)$ is constructed with multiple weighted normal distributions $g_j^i(x)$:

$$G^i(x) = \sum_{j=1}^{k} \omega_\ell g_\ell^i(x) = \sum_{\ell=1}^{k} \omega_\ell \frac{1}{\sigma_\ell^i \sqrt{2\pi}} e^{-\frac{(x-\mu^i)^2}{2\sigma_\ell^{i\,2}}}.$$

Such a PDF is easy to sample and provides a better flexibility to describe the landscape. Three elements decide the shape of the Gaussian kernel:

- ω_ℓ is the **weight** of the solution s_ℓ.

Solution Archive	Fitness	Weight
$[s_1^1 \ s_1^2 \ \dots \ s_1^i \ \dots \ s_1^d]$	$f(\mathbf{s_1})$	ω_1
$[s_2^1 \ s_2^2 \ \dots \ s_2^i \ \dots \ s_2^d]$	$f(\mathbf{s_2})$	ω_2
$[s_3^1 \ s_3^2 \ \dots \ s_3^i \ \dots \ s_3^d]$	$f(\mathbf{s_3})$	ω_3
$\vdots$	$\vdots$	$\vdots$
$[s_k^1 \ s_k^2 \ \dots \ s_k^i \ \dots \ s_k^d]$	$f(\mathbf{s_k})$	ω_k



$$G^i(x) = \sum_{l=1}^{k} \omega_l \frac{1}{\sigma_l^i \sqrt{2\pi}} \mathrm{e}^{-\frac{(x-\mu_l^i)^2}{2\sigma_l^{i\,2}}}$$

$$\omega_l = \frac{1}{qk\sqrt{2\pi}} \mathrm{e}^{-\frac{(l-1)^2}{2q^2k^2}}$$

$$\boldsymbol{\mu}^i = \{s_1^i, \dots, s_k^i\}$$

$$\sigma_l^i = \zeta \sum_{e=1}^{k} \frac{|s_e^i - s_l^i|}{k-1}$$

Figure 2.7: The solution archieve of ACO_R

- μ^i is i-th value of all the solutions in the archive.
- σ^i is standard deviation that determines that new samples.

Following are some parameters used by ACO_R that also effects the Gaussian kernel construction. k is the *size of the solution archive*, which works like the memory of the swarm. It may not be smaller than the number of dimensions. q represents the *locality* of the search process, *i.e.* how often should we not pick the best solution in the archive. It controls exploration so that give a higher value of q leads to slower yet more robust convergence. $\xi > 0$ represents the *speed of convergence* that has effect similar to the *pheromone evaporation* rate in ACO. It works like selection and enhance the good solutions in the archive.

The initial solution construction process proceed as following. After evaluating k randomly generates solutions, they are stored along with their fitnesses and weights in the solution archive. The weights are calculated according to the equation in Figure 2.7. Then the solution archive is sorted horizontally according to the fitness so that the best solution s_1 is on the top

For general *solution construction*, we first define m as the number of ants used in an iteration for sampling. During each iteration, m new solutions are constructed dimension by dimension by sampling the i-th dimension Gaussian kernel G^i A more detailed and efficient process of sampling the Gaussian kernel PDF is described in [18].

Pheromone update is accomplished by adding the new samples, their fitness, and the corresponding weight into the archive. Then, the solution archive is sorted according to the fitness and we only keep the top-k solutions in the archive. This selection process ensures only the best solutions are kept in the archive. Therefore, future samples will be constructed near these best solutions and guide the ants toward the optimum.

Chapter 3

Clustering Techniques

We are interested in solving real-valued multimodal problems composed of observable unimodals. Given the case, it is easier to solve an isolated unimodal within a subspace, than tackling the complete search space with the multimodal problem. Therefore, the first thing for solving a multimodal problem is to **identify** and **isolate** the potential unimodals within the given search space.

We tried to isolate potential *fitness hills*, *i.e.* unimodals, by **clustering** the initial samples points, and consider each cluster as a multi-dimension normal distribution. Different clustering techniques are often applied to identify different characteristics of clusters. Here we proposed a *hierarchical clustering* techniques to identify *fitness hills*, since we consider not only the density of the particles, but also the fitness values of different search points. Our basic assumption for the under lying unimodal is a weighted normal distribution, since we need to take fitness into account instead of viewing each point with the same weight. We tend to focus on the particles with better fitness, than a dense cluster with less fitness. It is also easier to calculate the weighted mean vector and weighted covariance matrix in higher dimension.

Later, we applied the **Minimum Description Length** (MDL) to reduce the number of clusters. Although a complex Gaussian Mixture Modal is able to describe the sample distribution better, we believe that a more compact model, in terms of information entropy, is the better choice when multiple models can describe the same distribution. This also allows us to define a more stable subspace for further searching.

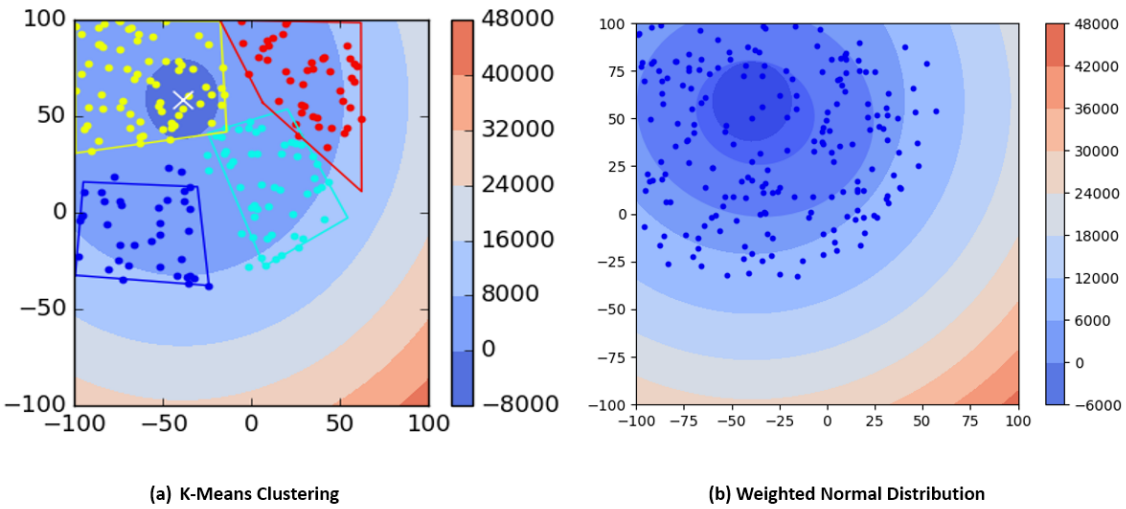
3.1 K-Means Clustering

In this section, we first describe the K-Means clustering techniques and its limitation for identifying the fitness hills. K-Means clustering aims to partition n observation data points into k clusters. There are two main procedures for K-Means clustering: the *assignment* step and the *update* step.

During the *assignment* step, an initial set of k means positions are given. Then, each data point is assigned to the cluster with the *nearest mean*. The assignment to the *nearest mean* can be formally described as creating clusters whose mean yields the least within-cluster sum of squares (WCSS), textiti.e. the sum of squared Euclidean distance. During the *update* step, the new centroids of each new clusters are calculated and assigned as the new means. This can be also be described as minimizing the WCSS. The algorithm proceeds by alternating between these two steps until the means no longer change. Since there only exists a finite states of partitions, the algorithm must converge to a local optimum. However, different initial mean positions results in different clusters.

One of the main problem for using the K-Means clustering to identify *fitness hills* is that it considers only the *spatial density* and does not utilize the fitness value of each particle. Different sets of initial centers might result in different kinds of clusters. Meanwhile, each cluster might be centering at a point that does not necessarily possess the best fitness in the neighborhood. Therefore, the K-Means clustering often results in **unstable clusters** depending on initial conditions, and is highly sensitive to particle density. Another common phenomenon for being sensitive to spatial density is that the points on the *border of clusters* might belong to different clusters after each update. This makes the clusters unstable and costs unnecessary evaluations and computation for redefining the borders of ROI after each update.

The other problem for using the K-Means clustering is that one needs to **decide the number of clusters**. Determining the number of clusters is also a highly studied subjects. We briefly discuss three common estimation methods that we have tried, the *silhouette coefficient* [17] , the *gap statistics* [20] , and the *Hartigans' dip test* [9] in the following paragraph.



(a) K-Means Clustering

(b) Weighted Normal Distribution

Figure 3.1: Comparing K-Means clustering with weighted Gaussian Distribution on CEC2005 F1 Problem

The *silhouette coefficient* measures how cohesive an object is to its own cluster and how separated it is to the other clusters. Ranging from -1 to 1, the silhouette coefficient with a higher value indicates it is more likely to belong to its own cluster. The *gap statistics* compares the change in within-cluster dispersion with that of an reference null distribution. It calculates the gap statistics for different number of clusters and select the one with the maximum gap statistics. The *Hartigans' dip test* uses the empirical cumulative distribution function (ECDF) to measure the maximum difference between multimodal samples and the unimodal samples. It calculates maximum deviation of the sample ECDF from the unimodal CDF and check if the difference is statistically significant.

We wish that as the algorithms proceed, particles should gather around the good solutions and gives a clear density signal. However, they seldom identify a unimodal at the beginning. It means that we need to add up more particles to meet the population size requirement for each clusters. This increases the density in that specific region, and creates an artificial cluster that should not exist in the first place.

Figure 3.1 shows how K-Means fails to identify a unimodal and results in four clusters due to density. A more preferable clustering result is shown on the right in Figure 3.1. We would like our clustering method to be capable of identifying the unimodal and be able to center around the particle with the best fitness.

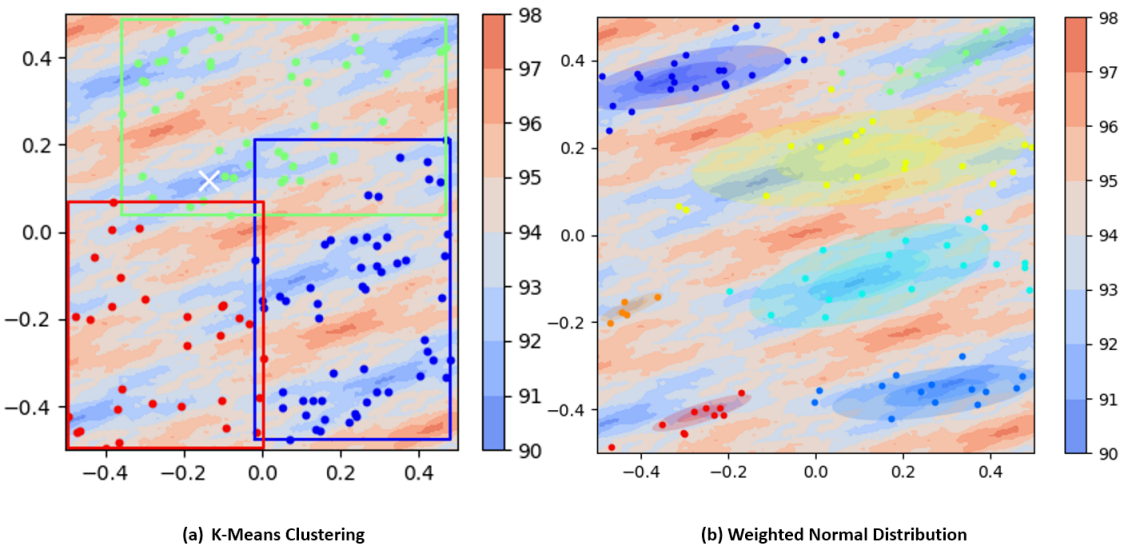


Figure 3.2: Comparing K-Means clustering with weighted Gaussian Distribution on CEC2005 F11 Problem

Figure 3.2 shows another example of how K-Means fails to identify the right multi-modals. Since K-Means only consider the density, it is hard to recognize the dense hills in the landscape. This initial clusters leads to more unnecessary evaluations for merging and re-identifying the hills. However, we can see that on the right, our technique successfully recognize almost all the hills in the problem. It also helps to locate the local optimum solutions approximately at the center. This gives lost of advantages for the algorithms to search, since the task becomes exploiting a single hill. Therefore, later we adopted the weighted normal distribution as the underlying model.

3.2 Heirarchical Clustering

Since we would like to recognize the *fitness hills*, we modified the agglomerative clustering technique to create hierarchical clusters. For a problem in D dimension, we defined the neighborhood of a particle to be the $4 + \lfloor 3\log(D) \rfloor$ particles with the minimum Euclidean distance, including the particle itself. Given the neighborhood of each particle, we create a directed graph by pointing each particle to the particle with the highest fitness in the neighborhood. If a particle does not possess the best fitness in its neighborhood, it points toward the direction with the maximum gradient. If it is the best particle in the

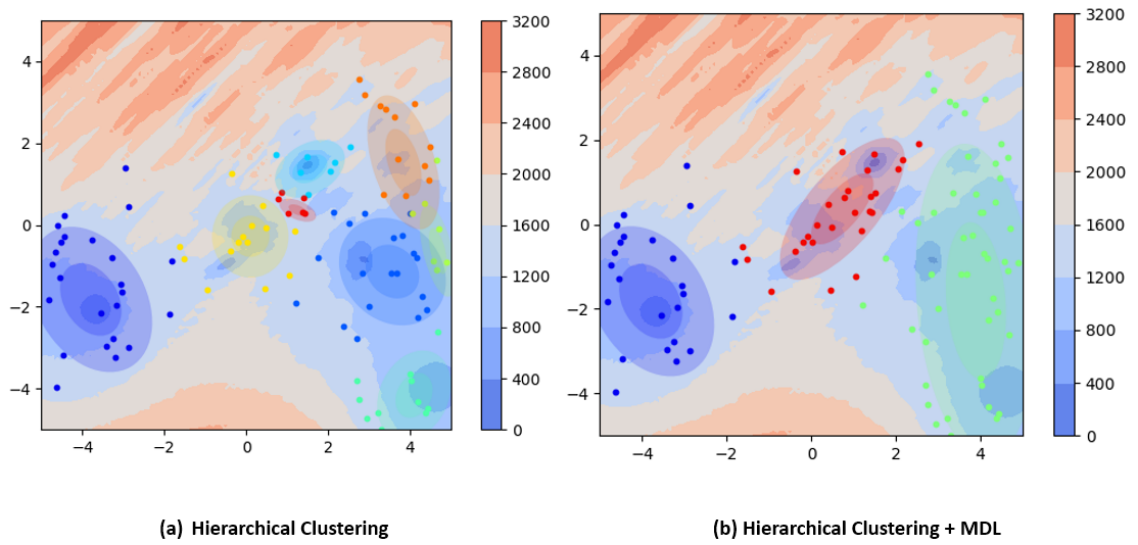


Figure 3.3: Before and after applying Minimal Description Length on CEC2005 F19 Problem

neighborhood, it is very likely to be at the approximate location of local minimum.

We then create trees with the directed graph, where each tree represents a cluster with a potential fitness hill. The best position within this cluster is indicated by the root node. This way, the fitness signal guides the construction of clusters, so that this method is able to identify more *fitness hills* than the K-Means clustering. Moreover, we do not need to provide the estimate number of clusters, since the agglomerative clustering method *intrinsically* identifies *fitness hills*. These characteristics solves some fundamental problems that we faced when using the K-Means clustering along with different number-of-clusters-estimation techniques.

We can see that for a multimodal problem depicted in Figure 3.3(a), the hierarchical clustering technique successfully identifies several fitness hills. However, for the unimodal problem in Figure 3.4(a), the hierarchical clustering creates multiple "ripples" on the marginal area. These clusters are created when the best particle is a bit further from the next best particle. This indicates that the using the Euclidean distance as the distance metric is not suitable for problems with larger hills.

In order to solve this problem, we assume the underlying hills are composed of multivariate normal distribution. Given the distribution and a sample position, we can use the *Manhanoblis distance* to serve as a more expressive metric. We would also like to

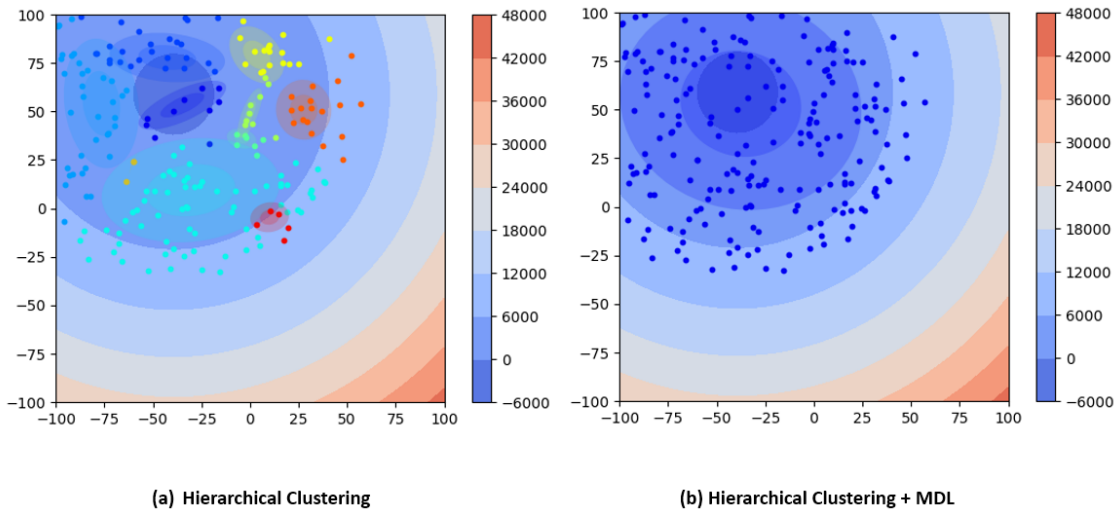


Figure 3.4: Before and after applying Minimal Description Length on CEC2005 F1 Problem

determine the number of clusters in the beginning, since we believe that the problems are composed of hierarchical hills. We do not need to invest so much resources in so many regions during the initialization. Reclustering after a few iterations shall give us a better signal about where to focus.

3.3 Minimum Description Length

The Minimum Description Length (MDL) principle was introduced by Rissanen in 1978 [14]. It adopts the idea of Occams's razor by selecting the codec that gives the shortest code length of the data. It provides a natural safeguard against overfitting, since it considers not only the goodness of the model, but also the complexity of the data to give the hypothesis. Here we modified the MDL described in [12] to help us determine the optimum number of clusters that provides the most efficient codes for the sample data. The MDL considers both *data description accuracy* and *model description efficiency*.

3.3.1 Data description accuracy

Since we would like to take the fitness into account, we use the weighted Gaussian distribution as the underlying model. For a cluster with N particles, each particle X_i has a rank r_i between 1 and N (ties are broken randomly) that corresponds to the fitness. The

weight ω_i can be defined as:

$$\omega_i = \log(N + 0.5) - \log(r_i). \tag{3.1}$$

The mean μ and covariance matrix Σ in a weighted Gaussian distribution N can be denoted as:

$$\mu = \frac{1}{\sum_{i=1}^{N} \omega_i} \sum_{i=1}^{N} \omega_i X_i, \tag{3.2}$$

$$\Sigma = \frac{1}{\sum_{i=1}^{N} \omega_i} \sum_{i=1}^{N} \omega_i (X_i - \mu)^T (X_i - \mu). \tag{3.3}$$

Let $X = \{X_1, X_2, ..., X_I\}$ denote the data set, composed of I samples. Each sample is a D-dimensional vector $X_i = \{X_{i1}, ..., X_{iD}\}$. The class-probability of observing sample X_i in class j, described by a parameter set Θ_j, can be denoted as $P_j(X_i|\Theta_j)$. The probability of observing the sample X_i in a finite mixture model with J components can be denoted as:

$$P(X_i|\Theta) = \sum_{j=1}^{J} \alpha_j P_j(X_i|\Theta_j), \tag{3.4}$$

$$\alpha_j = n_j / I, \tag{3.5}$$

where α_j is the prior probability that X_i belongs to the j-th model, and n_j denotes the number of samples in cluster j.

Therefore, the probability of observing sample X_i in cluster j can be further described by the j-th weighted Gaussain distribution:

$$P(X_i|\Theta_j) = N(X_i|\mu_j, \Sigma_j). \tag{3.6}$$

With the assumption that the data instance X_i are independently distributed, the joint data probability is the product of the individual instance probabilities:

$$P(X|\Theta) = \prod_{i=1}^{I} \sum_{j=1}^{J} \alpha_j P_j(X_i|\Theta_j). \tag{3.7}$$

However, we would like to give a higher weight for the data with better fitness. The

weight for the i-th particle that belongs to the j-th model with rank r_{ij} within the j-th cluster is:

$$\omega_{ij} = \log(n_j + 0.5) - \log(r_{ij}), \tag{3.8}$$

and the normalized weight is:

$$w_{ij} = \frac{\omega_{ij}}{\sum_{i=1}^{n_j} \omega_{ij}}, \tag{3.9}$$

the data description accuracy can be descrived as a weighted sum of individual log-likelihoods:

$$\log(P(X|\Theta)) = \sum_{j=1}^{J}(n_j(\log(\alpha_j) + \sum_{i=1}^{n_j} w_{ij} \log(P_j(X_i|\Theta_j)))). \tag{3.10}$$

3.3.2 Model description efficiency

Although a more complex model can describe the data distribution better, we believe that a simpler model with approximately the same accuracy is better. Therefore, the number of parameters that a model requires, is also considered in MDL. The free parameters in the Gaussian mixture model are:

- $J - 1$ paramters for J weights α_j, since $\sum \alpha_j = 1$
- D parameters for each mean μ_j
- $D(D + 1)/2$ parameters for each covariance matrix Σ_j

Therefore, the total number of free parameters is

$$\mathbb{K} = J - 1 + J(D + D(D + 1)/2) = J(D^2 + 3D + 2)/2 - 1. \tag{3.11}$$

Combining the *data description accuracy* and *model description efficiency*, the MDL defined in [15] is denoted as:

$$\min_{\mathbb{K},\Theta} - \log(P(X|\Theta)) + \frac{1}{2}\mathbb{K}\log(I). \tag{3.12}$$

3.3.3 Determining model with MDL

As mentioned before, we need to solve the problem of having too many "ripple" clusters on the margin area. Here we would like to use the MDL Equation to calculate model efficiency, and merge overfitting clusters. Therefore, after identifying potential unimodals with hierarchical clustering in Section 3.2, we use Equation 3.3.2 to calculate model efficiency. We created multiple weighted Gaussian distribution according to the hierarchical clusters, as shown in Figure 3.4(a) and Figure 3.3(a). Then we try all combinations of merging two clusters and calculate the corresponding MDL. Therefore, we compare the MDL score between the original hierarchical clusters and the new clusters with the minimal MDL score. If the MDL score of the new model is smaller than the previous model, we repeat the same procedure and keep merging. This procedure terminates when there is only one cluster remain, or when the minimum MDL score of all possible new modals is larger than the MDL score of the previous model. Then we get fewer yet still expressive clusters, shown in Figure 3.4(b) and Figure 3.3(b).

Chapter 4

Linear Projection

We would like to define a region of interest that contains only one unimodel for exploitation. Also, some algorithms, *e.g.* SPSO, requires a well-defined hypercube search space with fixed-value constraints in each dimension. Defining the projection onto subspace can be described as a model selection. In our case, this process does not require high accuracy, yet has a strict demand on time consumption. Combining the requirements, we use a linear projection matrix to project the original search space onto a subspace with feasible solutions bounded within $[0, 1]$ in each dimension.

One of the advantage for using projection matrix is that for a problem with ℓ variables, a linear projection matrix on homogeneous coordinate only requires $(\ell + 1)^2$ hyperparameters to be optimized. Therefore, less hyperparameters are assigned than directly define each hyperplane borders or vertices. This reduces the parameters that we need to optimize and results in less time consumption during model selection. We also use the (1+1)-ES, a fast and simple evolutionary strategy, to optimize the matrix. It allows us to rapidly approximate a high dimensional projection matrix within given number of iterations.

Furthermore, subspace projection also gives advantage for solving *inseparable problems*. For variables that are not independent, projection allows the algorithm to solve an easier, rotated and sheared problem on subspace, as shown in Figure 4.1. The homogeneous projection matrix separates the original ROI into less overlapped ROI, which reduces redundant search and sometimes enlarges the crucial regions.

In the following sections, we first describe the canonical affine transformation. Then

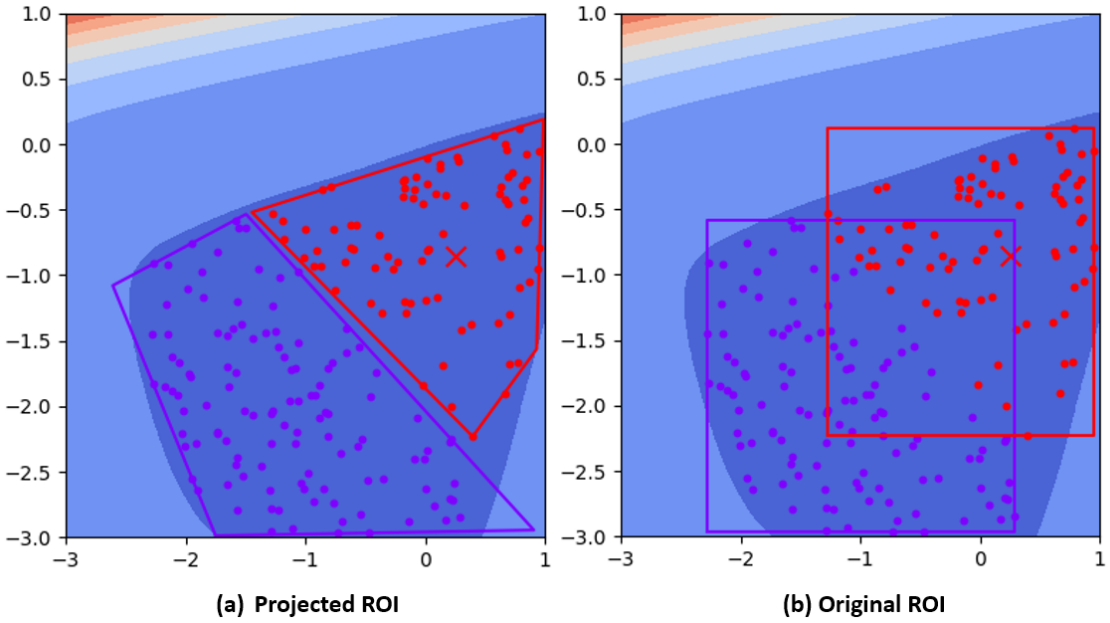


Figure 4.1: Projecting inseparable problems onto subspace.

we discuss how the projection matrix allows linear projection in a homogeneous coordinate. Finally, we give details of how we design the cost function and how we utilize the (1+1)-ES to optimize the projection matrix.

4.1 Affine Transformation

In geometry, an affine transformation preserves points, straight lines and planes. For two affine spaces A and B and any pair of points $P, Q \in A$, an affine transformation f determines a linear transformation φ that can formally defined as:

$$\overrightarrow{f(P)f(Q)} = \varphi(\overrightarrow{PQ}).$$

Here, we list some common 2D affine transformation matrix in Figure 4.2.

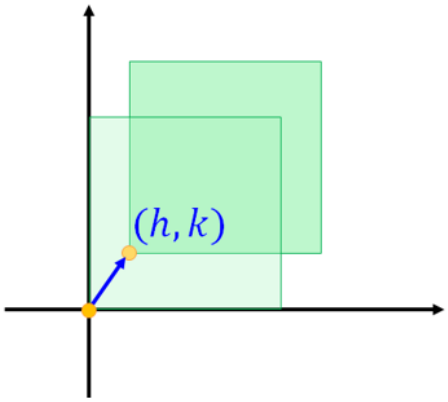
1. **Translation**

   Translation moves every point in the cluster by the same amount in a given direction. It allows us to reallocate the center of ROI to the positions with the best fitness.

2. **Rotation**

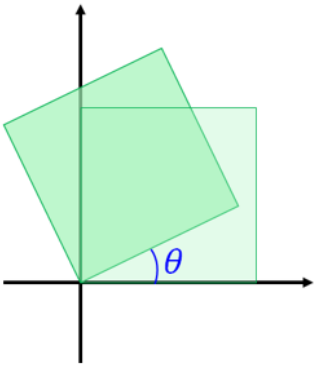
Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & h \\ 0 & 1 & k \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



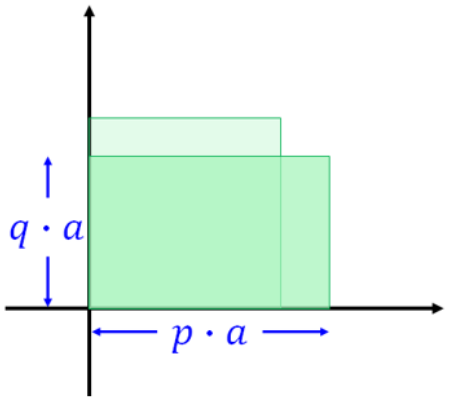
Rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Scaling

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} p & 0 & 0 \\ 0 & q & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Shearing

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \tan\varphi & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

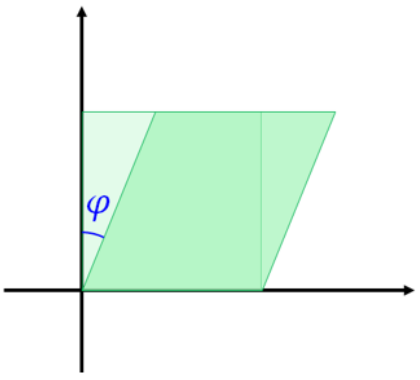


Figure 4.2: Some common affine transformation

Rotation is a common method to make a problem inseparable, by creating dependencies among variables. Therefore, using a rotation matrix to rotate the problem allows us to alter the problem characteristics. In higher dimensions, rotating about different axis requires different form of matrix. Here we only demonstrate a simple two dimensional case in Figure 4.2.

3. **Scaling**

   Scaling transformation can stretch or shrink a given search space. Since it changes the lengths and angles, scaling is not an Euclidean transformation. In search space projection, it allows us to enlarge certain areas that we are interested in, and shink some area that we tend to ignore on the subspace.

4. **Shearing**

   Shearing transform acts like "pushing" the search space in a direction parallel to a certain plane or axis. It changes the lengths and angles of certain area on the subspace. This transformation also allows us to expend certain interesting areas on subspace for a more thorough search.

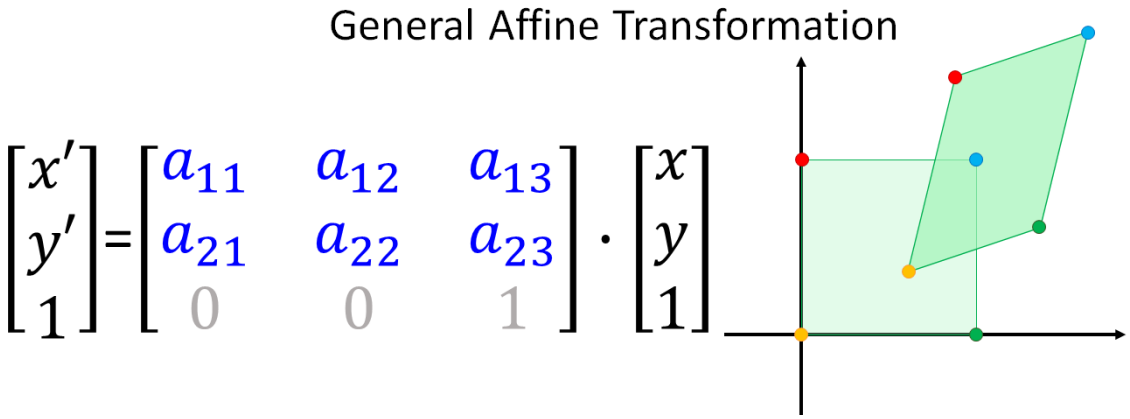


Figure 4.3: General affine transformation in 2D

The general affine transformation can be expressed as the form in Figure 4.3. All the common affine transformation matrix mentioned above, including translation, rotation, scaling and shearing can fit into this form. The general affine transformation can be seen

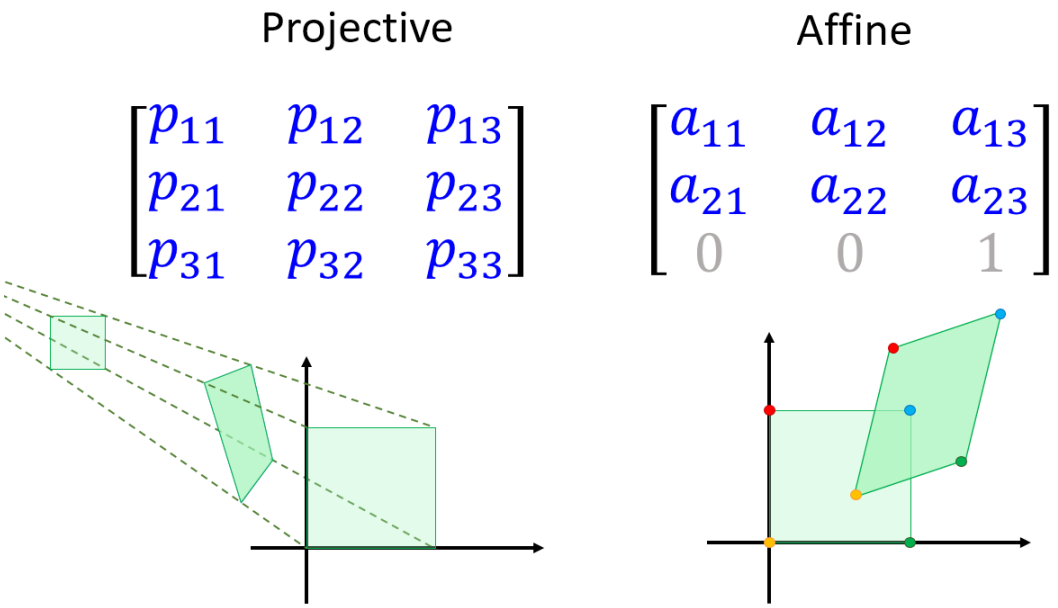


Figure 4.4: Affine transformation and projective transformation comparison.

as the matrix product of several affine transformation matrices. Although the affine transformation does not preserve the lengths and angles of certain shape, it does not alter the degree of a polynomial. That means parallel lines and planes are transformed to parallel lines and planes, and intersecting lines and planes are also transformed into intersecting lines and planes. It preserved certain geometric characteristics, which is beneficial for us to do searching on subspace.

However, the affine transformation can only create a hyperparallelepid ROI, as shown in Figure 4.4 The parallelogram shape is sometimes not flexible enough to define a non-overlapping ROI that matches the shape of the fitness landscape. In order to acquire a more flexible quadrilateral ROI, we need the projective transformation. Projective transformation is more expressive than affine transformation since the last row does not have to be 0, 0, and 1.

4.2 Projective Transformation

Projective transformations are the most general linear transformations and require the use of *homogeneous coordinates*. The *homogeneous coordinate* is also called *projective coordinates*, since it is a system of coordinates widely adopted in projective geometry. Given

Projective Transformation

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} \cdot z'$$

Figure 4.5: Projective transformation matrix.

Inverse Affine Transformation

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Inverse Projective Transformation

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}^{-1} \cdot \begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} = \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = \begin{bmatrix} x + e_x \\ y + e_y \\ 1 \end{bmatrix} \cdot z_r$$

Figure 4.6: Comparison between inverse affine transform and inverse projective transformation.

a point (x, y) on the Euclidean plane, than (xZ, yZ, Z) is called a set of homogeneous coordinates for the point, for any non-zero real number Z. One of the great advantage of homogeneous coordinate is that the points, including the ones at infinity, can be represented with simpler, finite coordinates. That is, when Z is 0, the homogeneous coordinates of the point is defined at infinity.

Therefore, as shown in Figure 4.5, for given a position (x, y) on the original search space, we get a projected homogeneous coordinate (x'', y'') on the subsapce. We can generalize this to any d-dimensional space. The *homogeneous convention* is to convert the projected position, a vector with $(d + 1)$ variables, to a d variable vector by dividing first d elements with the last element.

However, unlike the affine transformation, the projective transformation does not guarantee inverse transform. The projection matrix does not guarantee a non-zero matrix de-

terminant. Also, after inverse transform, there could be an error (e_x, e_y) that deviates from the original position (x, y), as shown in Figure 4.6. Therefore, we need to *optimize* the projection matrix in order to acquire a more suitable ROI, as well as small reconstruction errors for each samples.

4.3 Optimization for Projection Matrix

After identifying different clusters of samples, which represents a unimodal, we would like to further define a reasonably good ROI. In order to obtain a nice ROI for a problem with D variables, we need to optimize the $(D+1)^2$ parameters in the projection matrix with a loss function. The loss function should cost little computational time and should suggest the right direction for optimization, in order to match the restrictions of a well-defined ROI. Also, the optimization only needs to define a reasonably well ROI for the algorithms to search. In the later iterations, the matrix update procedure allows us to define a more accurate ROI with a better insight of the unimodal subproblem. In our case, it is acceptable to sacrifice a bit of accuracy for computational time in hyperparameters tuning. We will describe the design of our loss function and the advantage of using a (1+1)-ES to optimize the matrix in the following sections.

4.3.1 Loss function

The goal of the ROI is to separate the search space into non-overlapping subspaces, each containing an approximately normalized fitness hill. In our loss function, we considered the following features:

1. Distances to the boundary for the **points within the cluster** yet excluded in the ROI.

2. Distances to the boundary for the **random samples within other ROIs** yet included in the ROI.

3. Distances to the boundary for the **random samples** in the subspace that are outside of the original search space boundary.

4. Distance of the weighted **mean** to the center of subspace $[0.5]^D$

5. The difference for each element in the weighted **covariance matrix** to a scaled identical matrix

6. The sum of **reconstruction** error for each particle

We add up all the errors mentioned above and try to minimize that error. Following part explains the reason for considering each features.

First, we would like to keep all the particles that belong to this cluster in the boundary $[0, 1]^D$. It can easily be done by projecting all points within the cluster onto the subspace, and check whether if any of the projected position is outside of the boundary $[0, 1]^D$.

Second, we would like to avoid ROIs from overlapping by adopting the Monte Carlo method for region collision detection. When optimizing one ROI, we randomly generate a given amount of samples within the subspace for all the other ROIs. Then, we use the inverse projection matrix of each other ROIs to project these sample points back to the original search space. After that, we use the projection matrix of the ROI that we are optimizing to project all the exterior samples onto the subspace. This way, we can easily discover which exterior samples are within the $[0, 1]^D$ boundary. Figure 4.7 shows a screen shot of the red ROI before and after optimizing the projection matrix of the red ROI. We randomly generates 200 samples within the green and blue ROI and tries to exclude them in the red ROI.

Third, we also use the Monte Carlo method described before to make sure that the ROI does not exceed the boundaries of the orignal subspace. We randomly generate samples from the subspace and inverse project it back to the search space to make sure that all points are within boundaries. If not, we would like to minimize the distance of sample points to the nearest border.

Forth, we would also like each underlying model for each subproblem to be a multivariate Gaussian distribution with mean around the center $[0.5]^D$. If the algorithm searches near the boundary, it indicates that ROI should be updated since the optimum solution might be outside of the border. Placing the optimum solution at the center of the search

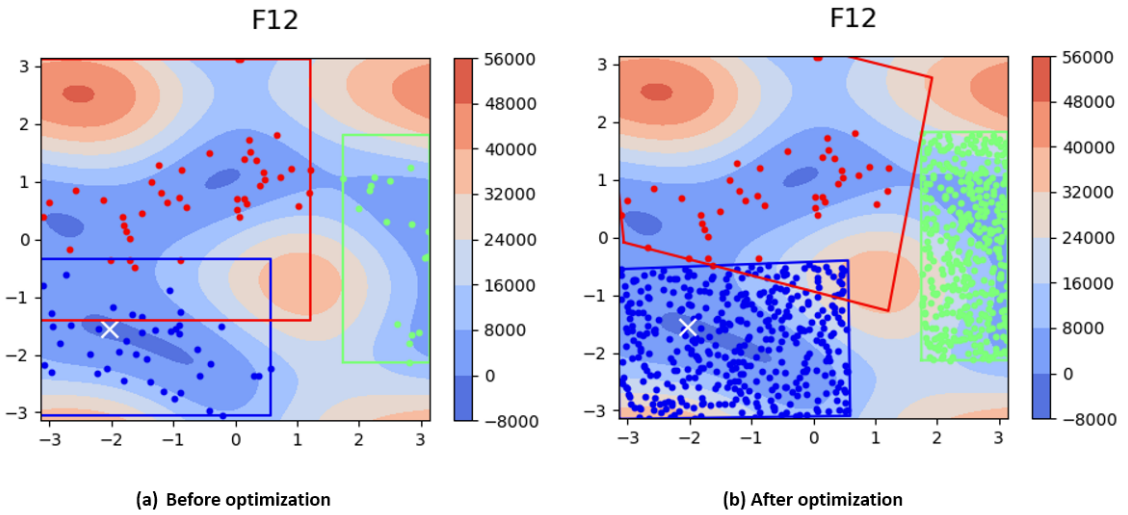


Figure 4.7: Snapshot of optimization for the projection matrix of the red ROI.

space creates a more stable model for the algorithm to search. This allows a thorough search around the center, and decreases the frequency of ROI update. Therefore, we calculate the weighted mean of the points within the cluster on the subspace. Then we minimize the distance of weighted mean to the center.

Fifth, we would like the weighted covariance matrix to be approximately $0.2I$, where I is an identical matrix. This enables the projection matrix to rotate, scale, and shear a fitness hill. The normalization preprocess is a common technique to make future optimization easier. Therefore, we calculate the difference for each element in the weighted covariance matrix and $0.2I$.

Finally, since the we project the particle positions in a homogeneous coordinate, the correlations between reconstructed positions on the subspace might not be identical to the original ones. We would like to minimize the transformation error while keeping the ROI within the original boundaries. Therefore, for each possible matrix solution, we use it to project all points in the corresponding cluster onto a subsapce. Later, we use the inverse matrix to project all the points back to the original search space to get the reconstructed positions. Then, we calculate the distance of all points in all dimensions between the original position and the reconstructed positions.

4.3.2 Optimization algorithms

With the loss function defined in the previous section, we still need an optimization algorithm to optimize the projection matrix. The difficulty for *hyperparameters optimization* is to balance between speed and optimization ability. We need the hyperparameters to define a subspace that contains the possible optimum solution while costing few computation time.

We can simplify and stabilize the optimization process by giving a fixed initial solution. Here, the initial solution is a simple dot product of a translation matrix and a scaling matrix. The matrix defines a hypercube, which is bounded by the minimum and maximum value in each dimension of all positions in the cluster. This allows the algorithm to start the search around a more preferred neighborhood.

We first tried CMA-ES to optimize the projection matrix. However, we found that it cost too much time to optimize the projection matrix with 961 variables in the 30-D problem Later, we utilize the (1+1)-ES described in Algorithm 1. We would like to reduce the computation time by giving the (1+1)-ES only one hundred iterations for each run of three random restarts. (1+1)-ES works well with one of the result shown in Figure 4.7

Finally, the infeasible solutions that belong to the cluster yet being outside of ROI after the optimization are repaired with the following steps. We normalize the distance between the infeasible solution and the center of ROI, to project it onto a hypershpere centering at $[0.5]^D$ and with radius 0.5. This way, although the reconstructed position is not accurate on the original search space, it is still able to indicate the approximate fitness in that direction in the subspace. Also, our basic assumption is a unimodal hill that puts the best fitness point in the center. Therefore, marginal points should have lower fitness and should not spend more resources to exploit. It just needs to give us a hint of the gradient in that direction.

Chapter 5

Multi-armed Bandit Algorithms

After identifying the unimodals in Chapter 3 and defining a ROI for exploitation in Chapter 4, we still need to decide how to **allocate our resources**. During different phases of searching, the *exploration vs. exploitation* dilemma needs to be handled accordingly. There have been many literatures considering how the algorithms converge and how to manipulate the searching step-size in order to find the global optimum more efficiently. Here, we propose that different strategies should be taken according to *evaluations left*. Generally, we would like to allow more exploration in the beginning when there are abundant evaluations left. Then, we would like to gradually increase the portion of exploitation behavior. When there are very few evaluations left, we should concentrate on exploiting the current best hill. Therefore, instead of letting the algorithms handle both exploration and exploitation, we propose a bandit technique to help manage the exploration, and leave the simpler subproblems for the algorithms to exploit.

Multi-armed Bandit (MAB) Algorithms are suitable for this scenario, since it learns model from outcomes and the actions it takes do not change the state of the world. Moreover, the decisions that it makes help discover more knowledge which can improve future decisions. This matches our description for exploring fitness landscape in real-valued optimization. However, our goal is a little bit different since we focus more about obtaining the optimum solution. Unlike canonical MAB algorithms that minimize regrets, we wish to maximize the probability of gaining the maximum rank.

In the following sections, we'll first describe the MAB problem. Then, we briefly

introduce some MAB algorithms that were described in the review which Vermorel et. al. made on MAB algorithms [21]. Finally, we propose a new bandit technique that aims to maximize the probability of gaining the maximum rank.

5.1 The Multi-armed Bandit Problem

Multi-aremed Bandit (MAB) Problem was originally described by Robins in 1985 [16]. In this problem, a gambler has to decide which machine to play in a row of slot machines, a.k.a one-armed bandits, and how often to play it. When the lever is pulled, each machine provides a reward according to a probability distribution. Therefore, the gambler iteratively plays one lever at each round and observes the probability of reward for each arms. Here, the gambler is also facing the *exploration vs. exploitation* trade-off. The problem of determining the best strategy for the gambler is called the Multi-armed Bandit problem.

The MAB problem can be more formally described as an agent deciding which one of the $K \in \mathbb{N}_+$ arms to pull at time t to receive the maximum reward. The K arms can be seen as a set of real distributions $B = \{R_1, ..., R_K\}$. Let $\mu_1, ..., \mu_K$ be the mean of rewards for each arm, and $\mu^* = \max_k\{\mu_k\}$ be the highest reward mean. The *regret* ρ after T rounds is defined as

$$\rho = T\mu^* - \sum_{t=1}^{T} r_t,$$

where r_t is the reward at time t.

The goal is to minimize the regret, which represents the expected difference between the total rewards of an optimal strategy, and the sum of the actual rewards that have been collected. The reason for concerning the regret is because we would like to achieve a *zero-regret strategy*. A *zero-regret strategy* is a strategy whose average regret per round ρ/T tends to zero with probability 1 when the number of played rounds tends to infinity. The *zero-regret strategy* are guaranteed to converge to an optimal strategy if enough of rounds are played [21].

5.2 Bandit Algorithms Overview

The ϵ-greedy strategy, first proposed by Watkins [22], is the simplest and one of the most widely used strategy to solve the MAB problem. Given an $\epsilon \in (0, 1)$, the ϵ-greedy strategy chooses a random lever with a probability ϵ, and chooses the lever with the highest estimated mean with a probability $1 - \epsilon$. It is also known as *semi-uniform* method since it implies a uniform probability of exploration among a set levers.

There are two variants of the ϵ-first strategy: the ϵ-first strategy and the ϵ-decreasing strategy. The ϵ-first strategy does pure exploration in the ϵT first rounds for a given $T \in \mathbb{N}$. Then for the remaining $(1 - \epsilon)T$, it does pure exploitation by pulling the lever with the highest estimated mean. Another approach is the ϵ-decreasing strategy. The ϵ greedy strategy is sub-optimal, since the factor ϵ prevents the strategy from getting arbitrarily close to the optimal lever. Therefore, the ϵ-decreasing strategy adopts a decreasing ϵ for getting arbitrarily close to the optimal strategy asymptotically [21]. By carefully choosing the ϵ, the ϵ-decreasing strategy can achieve zero regret.

The *SoftMax strategy*, also known as the Boltzmann Exploration, is first proposed in [13]. It chooses the k-th lever with probability

$$p_k = \frac{e^{\hat{\mu_k}/\tau}}{\sum_{i=1}^{n} e^{\hat{\mu_i}/\tau}},$$

where $\hat{\mu_i}$ is the estimated reward mean of the i-th lever, and $\tau \in \mathbb{R}_+$ is the *temperature* parameter.

The *Interval Estimation strategy* is first proposed in [10]. It estimates the *optimistic reward* of each lever within a certain confidence interval, and greedily choose the lever with the highest optimistic mean. The optimistic reward estimation asymptotically matches the true reward mean as the number of choosing that lever increases. The optimistic rewards of levers that are not frequently pulled are over-valued, and it leads to their exploration as the Interval Estimation strategy greedily choose the lever with the highest optimistic mean.

The Price of Knowledge and Estimated Reward (POKER) strategy [21] considers three

ideas: pricing uncertainty, exploiting the lever distribution and taking into account the horizon The *pricing uncertainty* is to assign a "value of information" to the knowledge acquired by pulling a particular lever. It is also referred as "exploration bonuses" in bandit literatures that helps balancing exploration and exploitation. The idea behind *exploiting the lever distribution* is that the properties of unobserved levers could be estimated from the observed levers. The *horizon* is the number of rounds remain to be played. Therefore, the ratio between exploration and exploitation should depend on the horizon. *Taking the horizon into account* can also help to estimate the *price* of the knowledge acquired in POKER.

5.3 The New Bandit Technique

We propose a new bandit technique that aims to maximize the probability of acquiring the best rank in the history. We only focus on ranks instead of real-valued fitness to generalize the case.

Let r be the probability of not being able to acquire the best point. Let m_i be the one of the possible models in the arm. Let D be the model of all the observed data. The we define the probability of not being able to acquire the best point given a set of observed data as:

$$P(r|D) = \sum_i P(r|D, m_i)P(m_i|D) = \sum_i P(r|m_i)P(m_i|D). \tag{5.1}$$

Therefore, we would like find a remain evaluation allocation that gives a minimum probability of not being able to aquire the best point.

In order to calculate $P(r|D)$, we need to calculate all possible models for an arm with given rank, total number of points in the search space, and the number of remain evaluations, as shown in Figure 5.1. The indigo area represents the observed model D, while the light blue area represents all the possible models given the number of remain evaluations. It means that if we invest the two remain evaluations on arm 1, we might discover two better positions with higher ranks, or two worse positions that gives fitness worse than the current rank 2 particle. That is why there are two possible positions on

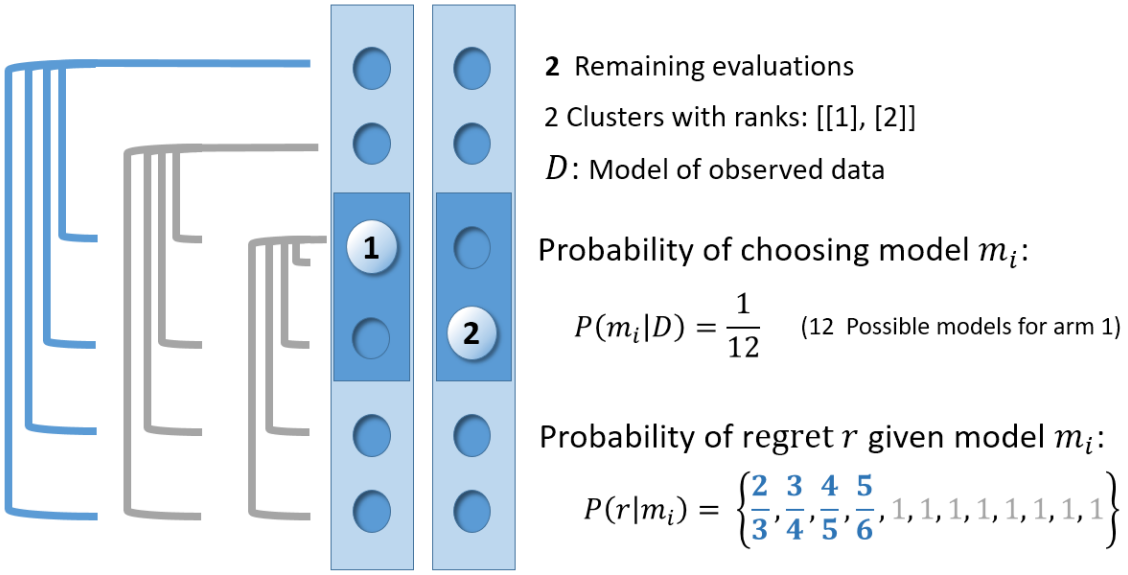


Figure 5.1: Probability of regret.

the top and two on the bottom of each arm. If the acquired fitness is between the current best and worst particle, than the rank would only exchange with another particle in one of the arms, so no more "holes" than the current ranks are needed. We can find 12 possible models that matches the current observation, *i.e.* rank 1 is included. Therefore, only 4 out of the 12 models are possible to acquire the best rank. We assume the model is uniformly distributed, that is, acquiring different ranks share the same possibility. We also assume that all models have the same probability to be chosen with the observed data model D. We can then calculate the probability of not being able to acquire the best rank in the clusters. To demonstrate the idea, we consider the right most blue model that contains the best and the worst possible ranks in Figure 5.1. The probability of not acquiring the best model for that specific model m_i is $P(r|m_i) = 5/6$, while the probability for choosing that model is $P(m_i|D) = 1/12$. Similarly, we can calculate the probability of regret for all models.

Given all the models and their corresponding probability of not acquiring the best rank, we can use Equation 5.3 to calculate the total probability of regret given a set of remain evaluations allocation. For example, if we decide to invest both of the remaining evaluations on the first arm, then the probability of not acquiring the best model for the right most model would be $P(r|m_i) = (5/6)^2$, while the probability for choosing that

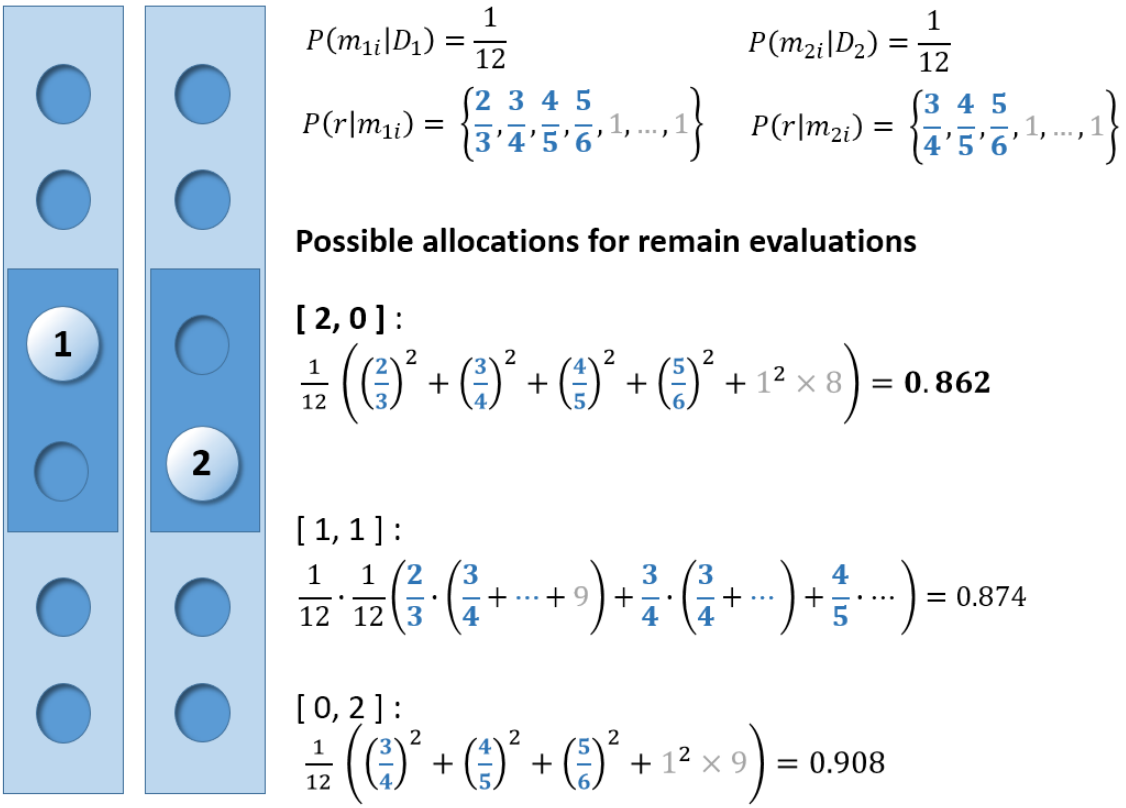


Figure 5.2: Probability of regret for all allocation combinations.

model is still $P(m_i|D) = 1/12$. Therefore, we can calculate the probability of regret as illustrated in Figure 5.2. After calculate the probability of regret all possible combinations, *i.e.* [2,0], [1,1], [0,2], we select the evaluation allocation model with the minimum regret. In this case, it is to invest all the remaining evaluations on the first cluster. This result satisfies our assumption that in the end with few evaluations left, one should focus on exploiting the current best hill. If today there are still lots of evaluations left, this technique gives an even evaluation allocation that only slightly favors the current best arm. This also satisfies our assumption that in the beginning, we should explore evenly between all possible clusters.

Chapter 6

The New Multimodal Optimization Technique

In this chapter, we illustrate the proposed multimodal optimization technique step-by-step. We first give the framework of the new multimodal optimization technique. Then we go through details about initialization and unimodal identification. We also discuss more about the implementation of optimizing projection matrix that defines the ROI. After that, we explain how to use the proposed multi-armed bandit technique during the optimization. Finally, we give details about the reclustering criteria, and some of the constraints of our techniques.

6.1 Framework of the New Multimodal Optimization Technique

The goal for our new technique is to identify the potential unimodals in the search space, define the ROIs for each unimodal for exploitation, and balance between exploration and exploitation through resource allocation. First, we need to modify the algorithms to meet some requirements of our technique. Each algorithm needs to be modified to satisfy the following conditions:

1. Able to update one individual at a time

2. Able to replace one individual with a given position and fitness

3. The algorithm can be projected onto a subspace and continue iterating

Since we consider resource allocation by pulling one arm at a time, we need to modify the implementation of algorithms to allow updating one particle at a time. Also, since we are defining the ROIs by projection, we need the same algorithm to continue after the ROI has changed. That means, given an updated projection matrix, we need to project all the necessary parameters for the algorithms onto to another subspace. Finally, after clustering, the number samples in one cluster might be more or less than the swarm size required by the algorithm. Therefore, we need to enable the algorithms to replace particles during optimization.

The new multimodal optimization technique starts with $100D$ samples within the given boundaries of a D-dimensional problem. We use selection pressure 2 to select particles with better fitness to identify potential unimodals in the search space. We use the hierarchical clustering technique and the MDL method mentioned in Chapter 3 to determine the clusters. Then, we define ROIs for each of these clusters with projection matrices, which are optimized by (1+1)-ES, as described in Chapter 4. After that, we initiate a designated algorithm for each arm on the subspace, initialized with the projected samples in each cluster. The arms are composed of a projection matrix, which defines the ROI, and an algorithm that is modified to allow projection, replacement, and update one particle procedures. Then we can use the new multi-armed bandit technique to determine resource allocation, *i.e.* which arm to pull. After each iteration, we need to check if the ROI of the latest pulled arm needs to be modified. We also check if the current cluster is possible to split into smaller clusters, or if any two existing clusters should be merged together. We'll go through the details of *reclustering* in the following paragraph. The algorithm terminates if the error reaches a predefined accuracy, or if the number of maximum evaluations is reached.

6.2 Initialization and Unimodal Identification

In the initialization process, we try to identify the unimodals on the given fitness landscape. In order to obtain a better resolution, we need a sample size that is large enough, and we need to perform selection after random initilization. A larger samples size ensures the sampling frequency is high enough to find hills with certain magnitude and width. We can always discover smaller hills later after the algorithms update the particles. Yet, it is better to find the right region to search in the first place to reduce unnecessary evaluations. The selection eliminates some noise for us to further identify the interesting regions.

We start by randomly sample $100D$ points in the given search space of a D dimensional problem. Then we select half of the particles with better fitness to get an indication of where the *fitness hills* might be. After that, we use the selected particles to identify potential unimodals, using the clustering technique mentioned in Chapter 3. We first perform a hierarchical clustering to consider not only the spatial density, but also the fitness gradients. It captures the characteristics of the landscape better than K-Means clustering, as explained in Chapter 3. Later we determine the number of clusters by MDL along with the underlying multivariate Gaussian model.

6.3 Define Region of Interest and Construct Arms

After locating the possible unimodals with the clusters, we can define the ROIs for each cluster by projecting the original search space onto a subspace that is only feasible within $[0, 1]^D$. We optimize each projection matrix with the (1+1)-ES and the loss function described in Chapter 4 We use the Monte Carlo method to avoid overlapping in higher dimension. That means, we randomly sample $100D$ points in each subspace $[0, 1]^D$ and project them back to the original search space. Then we project all the sampling points to one particular subspace that needs to be optimized, and try to exclude the samples from other clusters out of the feasible region $[0, 1]^D$ The projection matrices are optimized one-by-one, so the order might effect the optimization results. However, the initial clustering techniques provide a cluster approximate to a multivariate Gaussian distribution. There-

fore, the clusters are more likely to be in a "round" shape, that can be easily fit into a quadrilateral.

With the optimized ROIs and the samples in each clusters, we can initiate the arms for the new multi-armed bandit technique mentioned in Chapter 5 Each arm is composed of a **projection matrix**, that defines the ROI on the original search space, the mean μ and covariance matrix Σ of the underlying Gaussian model, and a **designated algorithm** with population size n, that performs optimization on the subspace. Notice that whenever an evaluation is needed for the algorithm, we project the position on the subspace back to the original space for evaluation. Therefore, the algorithm always conduct on the subspace without knowing the original search space. In order to initiate the algorithm, we need to modify the number of particles in each cluster to meet the required population. For clusters with samples more than n, we select the top-n particles with the best fitness. For clusters with samples less than n, we sample uniformly random points on the $[0, 1]^D$ subspace and inverse project it with the projection matrix back to the original search space to evaluate their fitness. Then we add them into the clusters and initiate the algorithm on the subspace.

6.4 Remain Evaluations Allocation and Recluster

In order to determine which arm to pull, we need to calculate the best remain evaluation allocation, as described in Chapter 5. We normalize the allocation vector and add it to a k-length record vector, where k is the total number of arms/clusters. During each iteration, we select the best arm, *i.e.* the arm that gives the highest value in the record, to update. We than minus the record value of that arm with 1, and ask the algorithm of the best arm to update one particle. Then we recalculate the remain evaluation allocation and add it into the record again. This way, the number of times each arm has been pulled should be asymptotically close to the time-variant resource allocation.

Right after each update, we check whether if we need to update the projection matrix to place the best point at the center of the ROI. This is done by a statistic test, which compares the weighted mean of the new samples against the original underlying multivariate normal model. Given a set of samples $X_1, ..., X_n \sim N_D(\mu, \Sigma)$ coming from a D-dimensional

normal distribution with μ and Σ known. For testing $H_0 : \mu = \mu_0$ versus $H_1 : \mu \neq \mu_0$, the statistic

$$Z = (\bar{X} - \mu_0)^T (\frac{\Sigma}{n})^{-1} (\bar{X} - \mu_0) = n(\bar{X} - \mu_0)^T \Sigma^{-1} (\bar{X} - \mu_0),$$

is used that follows $\chi^2(D)$-distribution under H_0. This is a generalization of the z-test, also known as u-test. We use this statistic test to examine if the new samples still comes from the same underlying multivariate normal distribution the ROI is constructed on. If the new samples is rejected by the statistic test that they do not come from the underlying distribution, we update the projection matrix of this specific arm to reallocate the ROI.

Whenever the ROI is updated, we also check if number of clusters should alter. This process is called *recluster*. We first need to check if the cluster in the current best arm can be split into multiple clusters with the hierarchical clustering and MDL techniques described in 3. Then, we check if any two clusters in the search space can be merged together using simply the MDL method mentioned in Chapter 3. This is a stable way to avoid "cracking" at the borders of different clusters. Also, whenever the recluster occurs, we reinitialize the record vector, since the clusters might be different from the last setup.

Chapter 7

Experiments

We use 25 benchmark problems from the CEC 2005 Special Session on Real-Parameter Optimization to test the performance of our new techniques. The CEC 2005 benchmark problems are composed of both standard test problems *e.g.* Sphere, Schwefel's, Rosenbrock's, Rastrigin's, etc., and some hybrid problems. We tested the CMA-ES, PSO, and ACO_R algorithms with and without our new techniques. The results show that our technique reduces the average error of CMA-ES and ACO_R on some multimodal problems.

7.1 CEC2005 25 Benchmark Problems

The CEC 2005 Special Session on Real-Paramter Optimization [19] aims to evaluate different algorithms in a more systematic manner by specifying a common termination criterion, size of problems, linkages/rotation, etc. It consists of 25 minimization problems with $2, 10, 30, 50$ dimensions. The evaluation requires 25 runs, each with maximum evaluations $10000 * D$. Initialization is required to be uniform random within the search space, except for problems 7 and 25, for which initialization ranges are specified. Similarly, the global optimum exists within the given bounds, except for problem 7 and 25. The algorithm is terminated one it reaches the maximum evaluations or if the minimum error in the function value is less than 10^{-8}. Here we only tested on the 2-dimensional problems, since our technique consumes too much time on higher dimensions, making optimization impractical. Also, we modified the Problem 11 and 12 by adding the Euclidean distance

to the bias position in the fitness function, so that there exists only one global optimum solution in the search space. Further speedup and studies to expend the technique on higher dimension are needed. We show the average and median error plots for all 25 problems in the following section.

7.2 Experiment Settings

In this section, we briefly describe the parameters setting for CMA-ES, SPSO and ACOR.

We use the *Covariance Matrix Adaptation Evolution Strategy for non-linear numerical optimization in Python* package [1] provided by Hensen, the author of CMA-ES. Although the default population for CMA-ES in D-dimension is set to be $4 + \lfloor 3\log(D) \rfloor$, meaning default population size is 6 in 2D, we found out that we a larger population for the MDL to not merge every particles into one cluster. Therefore, we set the population size as 30 for CMA-ES. The initial mean is set to be the center between max bounds and min bounds in every dimension. The initial step-size is set to be one sixth of the maximum length between max bounds and min bounds in each dimension. It is suggested to let the global optimum be within three standard deviation [6].

As for the SPSO 2011, there are currently no official Python implementation, so we created our own Python implementation according to the tutorials [4], and the official SPSO 2011 C++ code provided by Clerc, the author of SPSO [2]. The population size is set to be 40 as suggested in [4]. The parameters $c = \frac{1}{2} + \ln(2) \simeq 1.193$ and $w = \frac{1}{2\ln(2)} \simeq 0.721$ are also set as default. We implement the random topology with $K = 3$, and update the topology at the beginning and in every iteration when the best fitness does not improve. Also, we use the "bounce back" boundary condition suggested in [4], which is also described in Chapter 2.

For ACOR, we set our parameters according to the original paper [18]. The parameters are shown in Table 7.2.

[1] https://pypi.python.org/pypi/cma
[2] https://www.particleswarm.info

Parameter	Symbol	Value
No. of ants used in an iteration	m	2
Speed of convergence	ξ	0.85
Locality of the search process	q	10^{-4}
Archive size	k	50

Table 7.1: Summary of the parameters used by ACO_R

7.3 Experiment Results

Generally speaking, we can see a huge improvement in multimodal problems and hybrid problems when combining our technique with ACO$_R$, since the underlying model for ACO$_R$ is a Gaussian Kernel that intrinsically models hills. We can see improvement in some cases when combining our technique with CMA-ES. However, for SPSO, most of the results costs more evaluations to get to same error level. This is because the underlying model for SPSO is not a Gaussian, but a random topology. The flying particles make reclustering and redefining ROIs really unstable.

Problem 1 to Problem 5 are unimodal functions. The optimum solution of Problem 5 is on the diagonal of the search space, making it extremely easy to solve for SPSO and ACO$_R$. Thus there is no error line for these two algorithms on Problem 5 We can see that CMA-ES is extremely efficient on solving unimodal problems, since the underlying model is a multivariate Gaussian and it converges very fast. As expected, our methods to cost slightly more evaluations than the original algorithms on unimodal functions, since we sample more points in the beginning. Also, although our clustering methods recognize unimodals in most of the times, it is not good at handling narrow valleys, *e.g.* Problem 3, and starts to split into multiple clusters once falls into the valley with small gradient. This multiplication gets worse since the smaller the ROI becomes, the less gradient info there is. Thus, the clustering methods would try its best to generate "hills" from the plane with almost no gradient. This is the reason why our technique does not improve the original algorithms. it falls into the valley

Problem 6 to Problem 12 are basic multimodal functions. We can see significant improvement in Problem 11 and Problem 12 on algorithm CMA-ES and ACO$_R$. Besides, ACO$_R$ combined with our techniques leads with the minimum error in most of the prob-

lems. This means that the divide-and-conquer technique works well on problems with multiple hills. The narrow valley characteristics also appears in Problem 6, so we spend more evaluations than the original algorithms. One interesting thing happens in Problem 9, where our technique failed to improve CMA-ES on a Rastrigin Problem. We suspect that it is due not being able to recognize the small hills, since CMA-ES samples with a normal distribution. Therefore, even if particles did get to the top of different hills, the MDL method would still recognize them as one normal distribution. Thus, no reclustering was performed and the population converged to the wrong hills sometimes.

Problem 13 and Problem 14 are expended problems, and they are a bit too difficult for most of the algorithms.

Problem 15 to Problem 25 are hybrid composition functions. SPSO is suitable for these kinds of problems, since SPSO is good at exploring and takes a bit longer to converge. However, our methods enhanced the ability of ACO_R, making it compatible with SPSO, and even better sometimes.

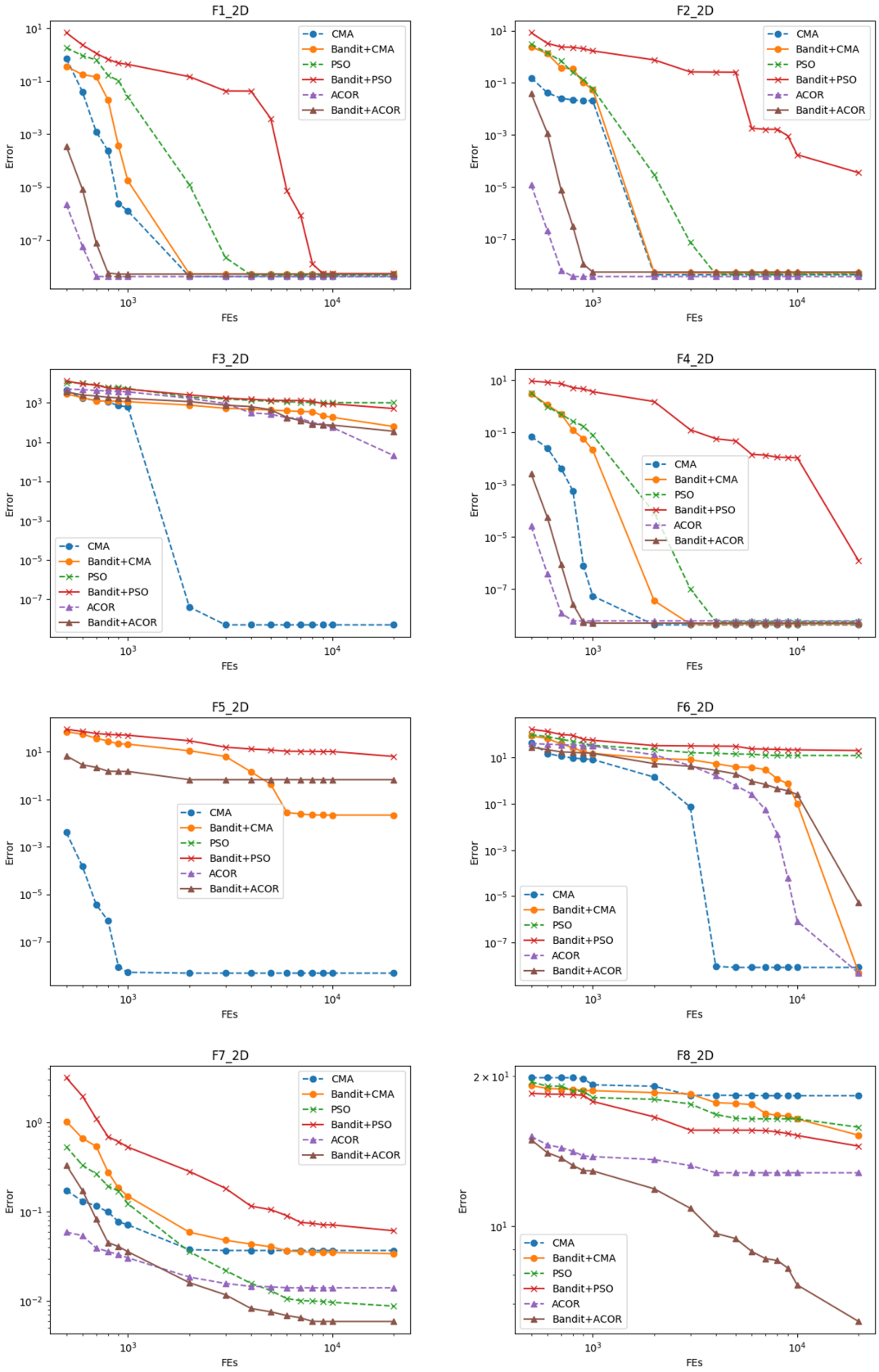


Figure 7.1: Average error of Problem 1 to Problem 8.

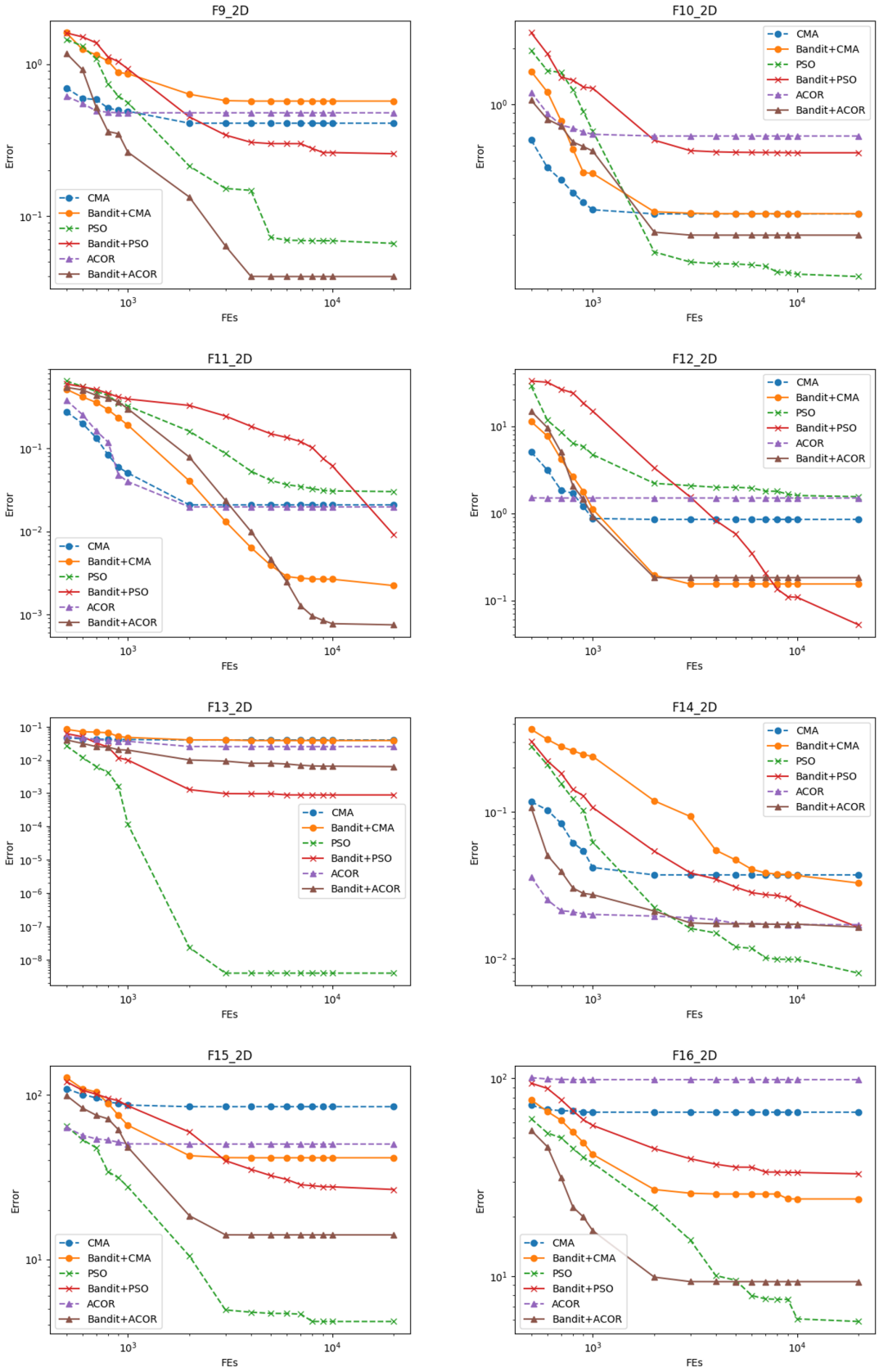


Figure 7.2: Average error of Problem 9 to Problem 16.

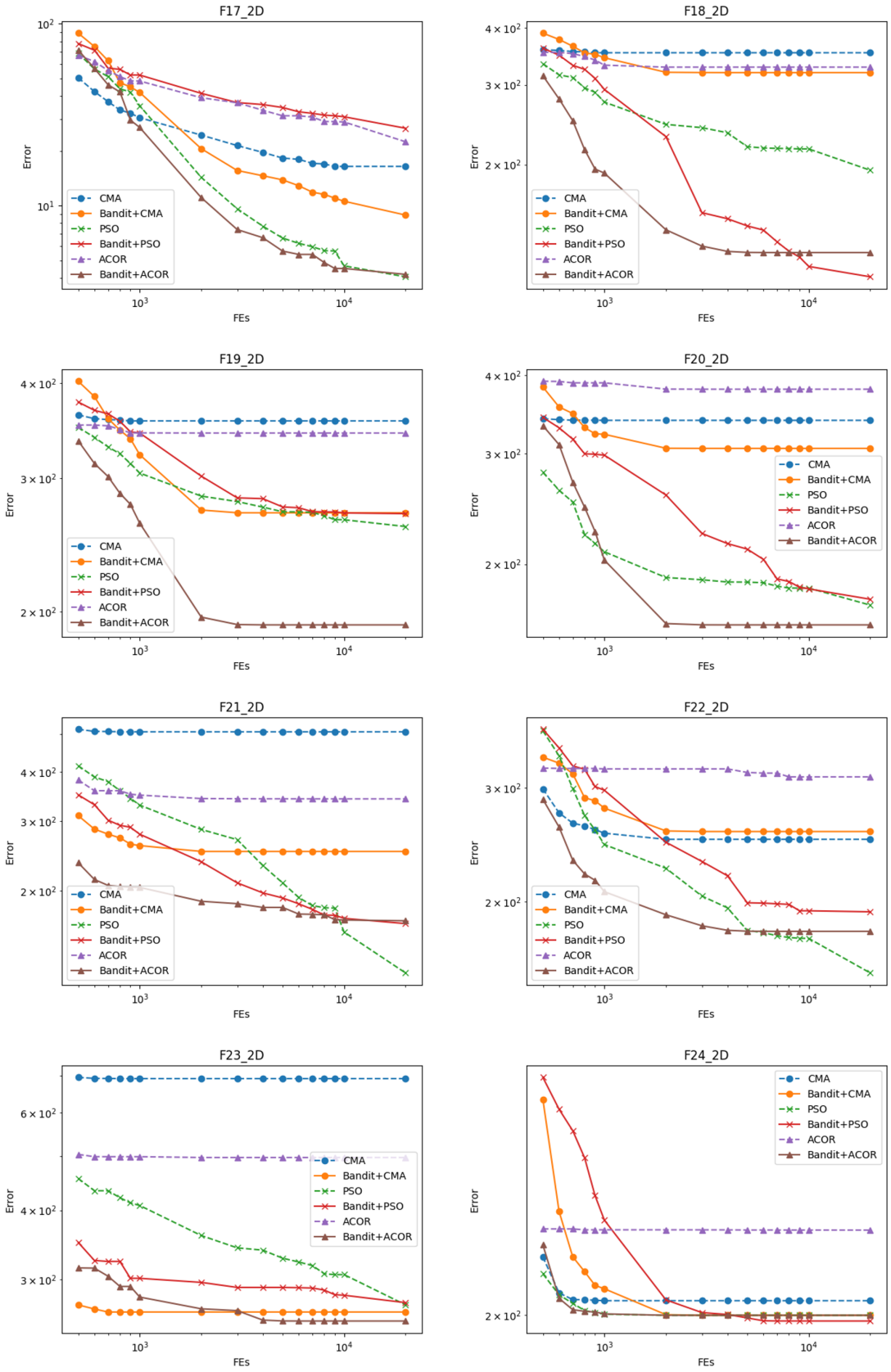


Figure 7.3: Average error of Problem 17 to Problem 24.

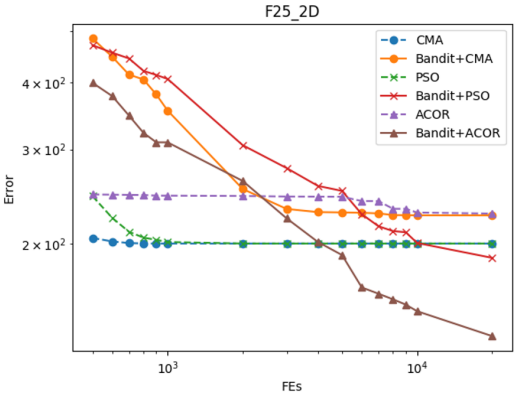


Figure 7.4: Average error of Problem 25.

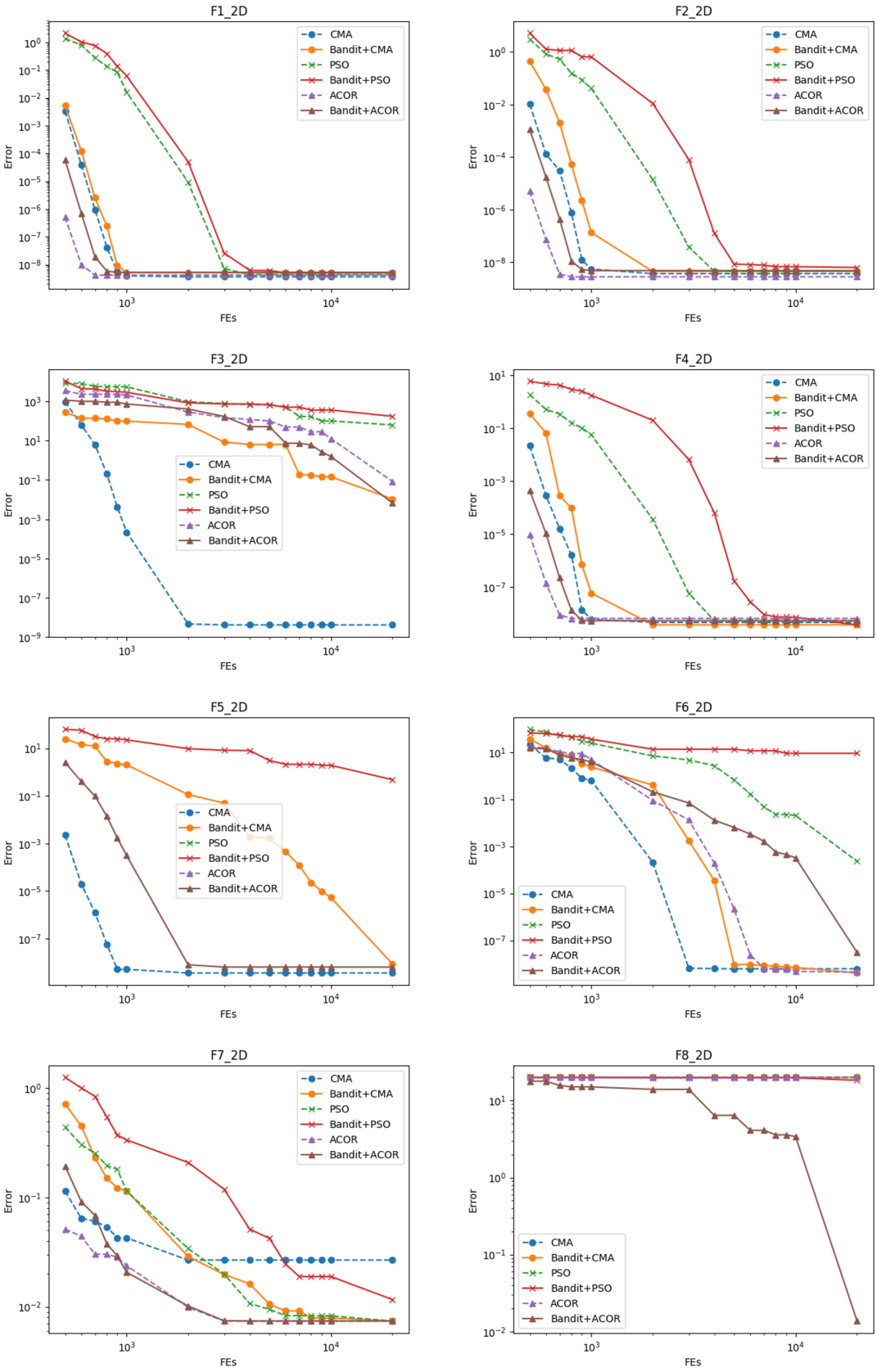


Figure 7.5: Median error of Problem 1 to Problem 8.

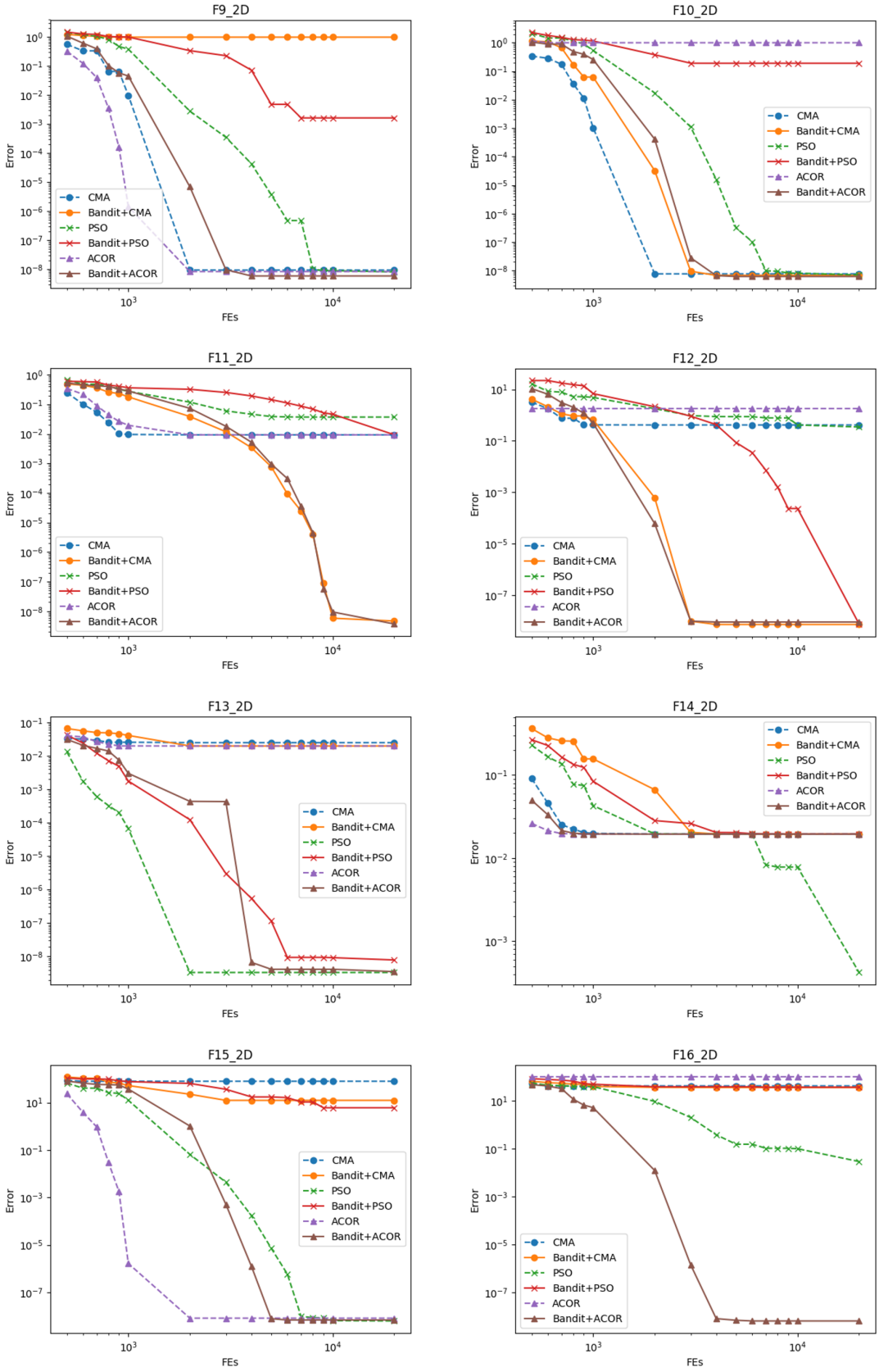


Figure 7.6: Median error of Problem 9 to Problem 16.

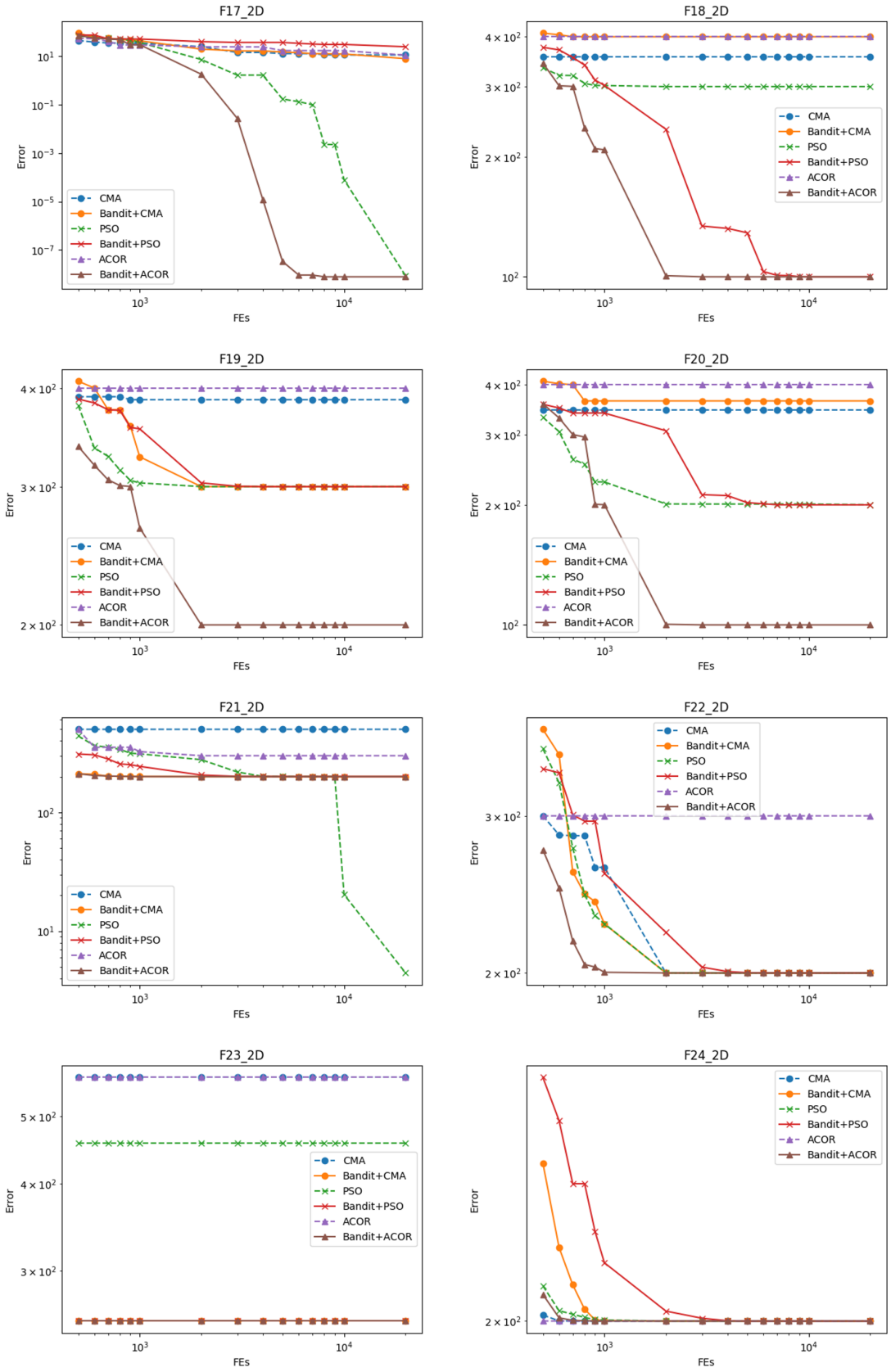


Figure 7.7: Median error of Problem 17 to Problem 24.

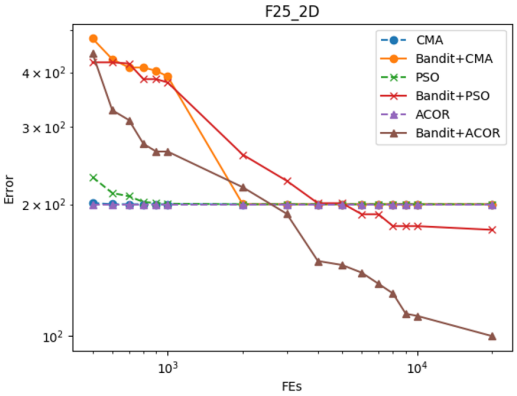


Figure 7.8: Median error of Problem 25.

Chapter 8

Conclusion

In this thesis, we proposed a new multimodal optimization technique that aims to break down the multimodal problem into unimodal problems. This technique is composed of three main components: unimodal detection, region of interest optimization and resource allocation. We use the three techniques to dynamically isolate potential unimodals and enlarge a specific interesting region on the subspace for algorithms to search more thoroughly. We also manage the ratio between exploration and exploitation according to the remaining resources, so that the optimization process can be more efficient.

First, we proposed a unimodal detection technique to find potential *fitness hills* in a real-valued multimodal problem by hierarchical clustering and Minimum Description Length. It depicts the underlying *fitness hills* better than K-Means. Also unlike K-Means which needs to collaborate with other methods to determine the number of clusters, our technique intrinsically detects the number of potential unimodals.

Second, we also proposed a method to separate the search space into smaller subspaces in order to enhance performance. We use linear projection matrix to project the search space to a subspace with well-defined boundaries for feasible solutions. Therefore, the algorithm only needs to search within a hypercube, constrained by $[0, 1]$ in all dimensions. This technique is preferable for algorithms that requires a box-shape boundaries. It also creates ROIs on the original search space that isolates the unimodals. We also use the (1+1)-ES to optimize the projection matrix so that the ROIs have minimal-overlapping, thus enhance the searching efficiency. By cracking down the original search space into

multiple non-overlapping subspace, algorithms now only have to search a smaller portion of the landscape which ideally consists of only one unimodal.

Furthermore, we proposed a new Multi-armed Bandit techniques that optimize the resource allocation. We also explained how to maintain a more stabilize cluster shapes during *recluster*, yet still conserve the flexibility to split when needed. We believe that in the beginning, when there are abundant of resources left, we should invest more resources in exploration. That means we should update each clusters more equally for exploration. Later, as the algorithms update the particles, we should be able to merge clusters to eliminate redundant search, or split a cluster and invest more particles to search in a certain region. When there are few evaluations left, we should concentrate on exploiting the current best hill.

However, there are still many issues that can be improved. Our experiment results shows that there are still stability issues. We need to investigate more on how to delete unnecessary arms. Thus, this technique tends to cost more evaluations on problems that need to move for a long distance after converging to a relatively narrow valley. Also, the clustering techniques to identify underlying unimodals can be further improved. The current hierarchical clustering technique can be extended to a complete tree that contains all clusters. Moreover, since this technique is composed of many relatively complex methods, speeding up computational time is also a crucial issue for all hyperparameters tuning techniques.

Bibliography

[1] A. Auger. Benchmarking the (1+1) evolution strategy with one-fifth success rule on the bbob-2009 function testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2447–2452. ACM, 2009.

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

[3] M. Clerc. Back to random topology. *Online at http://clerc. maurice. free. fr/pso*, 2007.

[4] M. Clerc. Standard particle swarm optimisation. *Online at https://hal.archives-ouvertes.fr/hal-00764996/*, 2012.

[5] M. Dorigo and G. Di Caro. Ant colony optimization: a new meta-heuristic. In *Evolutionary Computation, 1999. The 1999 IEEE Congress on*, volume 2, pages 1470–1477. IEEE, 1999.

[6] N. Hansen. The cma evolution strategy: a comparing review. *Towards a new evolutionary computation*, pages 75–102, 2006.

[7] N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18, 2003.

[8] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[9] J. A. Hartigan and P. Hartigan. The dip test of unimodality. *The Annals of Statistics*, pages 70–84, 1985.

[10] L. P. Kaelbling. *Learning in embedded systems*. 1993.

[11] J. Kennedy and R. E. P. S. Optimization. Ieee int. 4, 1995.

[12] I. O. Kyrgyzov, O. O. Kyrgyzov, H. Maître, and M. Campedel. Kernel mdl to determine the number of clusters. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 203–217. Springer, 2007.

[13] R. D. Luce. *Individual Choice Behavior a Theoretical Analysis*. 1959.

[14] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[15] J. Rissanen. Universal coding, information, prediction, and estimation. *IEEE Transactions on Information theory*, 30(4):629–636, 1984.

[16] H. Robbins. Some aspects of the sequential design of experiments. In *Herbert Robbins Selected Papers*, pages 169–177. 1985.

[17] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[18] K. Socha and M. Dorigo. Ant colony optimization for continuous domains. *European journal of operational research*, 185(3):1155–1173, 2008.

[19] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-p. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. *KanGAL report*, 2005005:2005, 2005.

[20] R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[21] J. Vermorel and M. Mohri. Multi-armed bandit algorithms and empirical evaluation. In *ECML*, volume 3720, pages 437–448. Springer, 2005.

[22] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[23] M. Zambrano-Bigiarini, M. Clerc, and R. Rojas. Standard particle swarm optimisation 2011 at cec-2013: A baseline for future pso improvements. In *Evolutionary Computation, 2013. The 2013 IEEE Congress on*, pages 2337–2344. IEEE, 2013.