A Study of

Attraction Basin Sphere Estimation

for Niching Evolutionary Algorithms

Xu Zhuoran

Doctor of Philosophy

Division of Synergetic Information Science

Graduate School of Information Science and Technology

Hokkaido University

January 2015

Contents

1 Introduction **1**

1.1 Background and Research Objective 1

1.2 Mathematical Optimization 4

1.2.1 Fitness Landscape 6

1.3 Evolutionary Algorithm 8

1.3.1 Genetic Algorithm 9

1.3.2 CMA-ES 11

1.3.3 Neuroevolution 13

1.3.4 Particle Swarm Optimization 14

1.3.5 Differential Evolution 16

1.3.6 Variable Mesh Optimization 16

1.4 Multimodal Optimization 17

1.5 The Existing Niching Techniques 17

1.5.1 Radius-based Niching Methods 18

1.5.2 Detect-Multimodal Method Based Niching Methods . . 24
1.5.3 Crowding Methods 25
1.5.4 Other Niching Methods 26
1.6 The Problems or Challenges in the Existing Niching Techniques 27
1.7 Conclusion 28

2 Attraction Basin Sphere Estimation 29

2.1 Introduction 29
2.2 Attraction Basin Sphere 30
2.3 Detect-multimodal Method 33
2.4 The Estimation Method 34
2.5 Examples and Discussions 36
2.6 Conclusion 41

3 Attraction Basin Sphere Estimation Genetic Algorithm 42

3.1 Introduction 42
3.2 Topological Species Conservation Algorithm 43
3.3 Attraction Basin Sphere Estimation Genetic Algorithm 45
3.3.1 Overview 45
3.3.2 Seed Update 45
3.3.3 Seed Conservation 50
3.3.4 ABSEGA Algorithm 51

3.4 Experiments 53
3.4.1 Test Function 53
3.4.2 Performance Criteria 55
3.4.3 Performance Comparison 57
3.4.4 Computational Cost 59
3.4.5 Internal analysis of ABSEGA 62
3.4.6 Visualization 65
3.5 Conclusion 66

4 Attraction Basin Sphere Estimation Genetic Algorithm for Neuroevolution 67

4.1 Introduction 67
4.2 Problems of ABSEGA in High Dimensional Space 69
4.3 Seed Importance Calculation 70
4.4 ABSEGA2 Algorithm 73
4.5 Experiments 75
4.5.1 Benchmark Tests 75
4.5.2 Robot Arm 77
4.6 Conclusion 84

5 Attraction Basin Sphere Estimation CMA-ES 86

5.1 Introduction 86

5.2 Seed Importance Calculation2 87
5.3 Attraction Basin Sphere Estimation CMA-ES 89
5.4 Experiments 92
5.4.1 Internal Analysis of ABSE-CMA-ES 96
5.4.2 Comparative Study With Others Algorithms 104
5.4.3 ABSEGA, ABSEGA2 and ABSE-CMA-ES 106
5.4.4 Visualization 110
5.5 Conclusion 112

6 Conclusion **113**

Acknowledgment **116**

List of Figures

1.1 An example of fitness landscape 7

1.2 An example of peaks 8

1.3 An example of global and local peaks 8

1.4 Artificial neural network 15

1.5 An example of standard genetic algorithm 22

1.6 An example of dynamic fitness sharing 22

2.1 An example of attraction basin 30

2.2 An example of multimodal 31

2.3 An example of unimodal 32

2.4 An example of ABS 33

2.5 Two given points are detected unimodal 34

2.6 Two given points are detected multimodal 34

2.7 An example of attraction basin sphere estimation 38

2.8 An example of an attraction basin covered by several ABS . . 39

2.9 An example of how ABSE fixs mistakes 39

3.1 An example of cover 46

3.2 TR of TSC and ABSEGA to reach the same level 61

3.3 Box plot of result grouped by sampling points on F1(left), F2(right) 63

3.4 Box plot of result grouped by sampling points on F3(left), F4(right) 63

3.5 Box plot of result grouped by sampling points on F5(left), F6 2D(right) 64

3.6 Box plot of result grouped by sampling points on F6 10D(left), F7(right) 64

3.7 F1(left) and F2(right) results visualized 65

3.8 F3(left) and F4(right) results visualized 66

4.1 Difference between high and low dimensional space 71

4.2 Big and small valleys 71

4.3 How to calculate importance 73

4.4 The ball catching task 77

4.5 Fitness of 8 algorithms 82

4.6 Diversity of 8 algorithms 83

5.1 An example of seed importance calculation2 89

5.2 Fitness of three algorithms on the ball catching task 109

5.3 Diversity of three algorithms on the ball catching task 110

5.4 F72d (left) and F82d (right) on 0,100,...300 generation 111

List of Tables

3.1 Mutation rate 59

3.2 Criteria value of ABSEGA and TEC 59

3.3 Computational cost to reach three levels 61

3.4 Discrete points for sampling 62

3.5 Mean and standard deviation of result of randomly sampled settings 63

4.1 PR of benchmark tests 76

4.2 PA of benchmark tests 76

4.3 DA of benchmark tests 77

4.4 Feature vectors of an individual 81

4.5 Feature vectors of all individuals in the population 81

4.6 The contents of the set D 81

4.7 Diversity at different levels 84

5.1 Test Functions 94

5.2 MaxFEs of test functions 95
5.3 Parameters for ABSE-CMA-ES 99
5.4 Results of PR of different parameter settings 99
5.5 Results of SR of different parameter settings 100
5.6 Results of Friedman's Test of different parameter settings . . 100
5.7 Holm's Test for $\alpha = 0.05$ of different parameter settings 100
5.8 Results of Wilcoxon's Signed-Rank Test of different parameter settings 101
5.9 Results of PR of disabling different components of the algorithm 102
5.10 Results of SR of disabling different components of the algorithm 103
5.11 Results of Friedman's Test of disabling different components of the algorithm 103
5.12 Holm's Test for $\alpha = 0.05$ of disabling different components of the algorithm 103
5.13 Contrast estimation of disabling different components of the algorithm 104
5.14 Results of PR of the comparison with other algorithms 105
5.15 Results of SR of the comparison with other algorithms 105
5.16 Results of Wilcoxon's Signed-Rank Test of the comparison with other algorithms 106
5.17 Results of PR of three algorithms 107

5.18 Results of SR of three algorithms 108

5.19 Results of Friedman's Test of three algorithms 108

5.20 Holm's Test for $\alpha = 0.05$ of three algorithms 108

5.21 Contrast estimation of three algorithms 109

5.22 Diversity of three algorithms at different levels 109

Chapter 1

Introduction

1.1 Background and Research Objective

In mathematics and computer science, an optimization problem is the problem of finding the best solution from all feasible solutions. However, knowledge of multiple solutions to an optimization task is especially helpful in engineering. For example, due to physical (and/or cost) constraints, the best results may not always be realizable. Multimodal optimization deals with tasks that involve finding all or most of the multiple solutions as opposed to a single best solution. Evolutionary Algorithms (EAs), benefit by being population-based, has the potential ability to locate multiple solutions. However, this is against the natural tendency of EAs, which will always converge to the best solution, or a sub-optimal solution. Finding and maintenance of

multiple solutions is wherein lies the challenge of using EAs for multimodal optimization.

Niching is the technique that can help EAs to find and preserve multiple stable niches, or favorable parts of the fitness landscape possibly around multiple solutions, so as to prevent convergence to a single solution. However, most niching methods employ a radius parameter which is hard to correctly set in practice.

The purpose of the study is to propose a new niching evolutionary algorithm (the combination of a niching method and an EA), which have the following new features:

1. The algorithm does not use the radius parameter.

2. The algorithm can create niches adaptively according to the fitness landscape.

3. The algorithm should overcome other niching methods on the ability of finding multiple solutions.

4. The algorithm should be efficient in computational time.

This thesis is organized into six chapters. In Chapter 1, I introduce the background and purpose of this thesis. I first give a review of Mathematical Optimization and Evolutionary Algorithms. Second, I introduce the concept

of Multimodal Optimization. After that, I explain the mechanism of Niching Evolutionary Computation and introduce several classic niching methods. Finally, I conclude the problems and challenges in the existing niching techniques, and describe the motivation of this study.

In Chapter 2, I propose a niching method named as Attraction Basin Sphere Estimation (ABSE). Given several candidate solutions, this method can collect niching information about those candidates from fitness landscapes and adaptively adjust niching parameters. This method can be coupled with a lot of EAs.

In Chapter 3, I apply ABSE to genetic algorithms to create a niching genetic algorithm: Attraction Basin Sphere Estimation Genetic Algorithm (ABSEGA). It identifies optimal-like individuals (seeds) from the population, and uses ABSE to calculate niching parameters. The experiments are performed on benchmark tests. The performance and efficiency are discussed.

In Chapter 4, I improve ABSEGA for Neuroevolution problems. The original ABSEGA does not work well on Neuroevolution problems, because those problems usually have a lot of local optimal solutions and ABSEGA may find multiple local optimal solutions instead of multiple global optimal solutions. I propose a method to calculate the Importance of solutions. Importance measures the possibility of a solution to be a global optimal solution. So the algorithm can find multiple global optimal solutions. The newly proposed

method is named as ABSEGA2. I examine this method in a robotic arm problem, in which I evolve an artificial neural network (ANN) to control the arm to catch balls. I also compare ABSEGA2 with other state-of-the-art algorithms.

In Chapter 5, I apply ABSE to Covariance Matrix Adaptation Evolution Strategy (CMA-ES), a very powerful local optimization method. The proposed niching method is named as Attraction Basin Sphere Estimation CMA-ES (ABSE-CMA-ES). The experiments is performed on a benchmark set provided by CEC 2013 Niching Methods Competition.

Finally, I conclude this thesis and discuss future researches in Chapter 6. For the remainder of the thesis, all optimization problems are assumed to be maximization problems.

1.2 Mathematical Optimization

A mathematical optimization problem, or just optimization problem, has the form

$$\begin{aligned} &maximize\ f(x) \\ &subject\ to\ g_i(x) \leq b_i, i = 1, ..., m. \end{aligned} \tag{1.1}$$

Here the vector $x = (x_1, ..., x_n)$ is the optimization variable of the prob-

lem, the function $f : R^n \to R$ is the objective function (fitness function), the functions $g_i : R^n \to R$, $i = 1, \ldots, m$, are the (inequality) constraint functions, and the constants $b_1, \ldots, b_m$ are the limits, or bounds, for the constraints. A vector x^* is called optimal, or a solution of the problem 1.1, if it has the biggest objective value (fitness) among all vectors that satisfy the constraints: for any z with $g_1(z) \leq b_1, \ldots, g_m(z) \leq b_m$, we have $f(z) \leq f(x^*)$ [4].

The optimization problem is an abstraction of the problem of making the best possible choice of a vector in R^n from a set of candidate choices. The variable x represents the choice made; the constraints $g_i(x) \leq b_i$ represent firm requirements or specifications that limit the possible choices, and the objective value $f(x)$ represents the utility of choosing x. A solution of the optimization problem corresponds to a choice that has maximum utility among all choices that meet the firm requirement [4].

Optimization problems exist widely in mathematics, computer science, economics, management science, and other fields. An example is device sizing of electronic design, which is the task of choosing the width and length of each device in an electronic circuit. Here the variables represent the widths and lengths of the devices. The constraints represent a variety of engineering requirements, such as limits on the device sizes imposed by the manufacturing process, timing requirements that ensure that the circuit can operate reliably

at a specified speed, and a limit on the total area of the circuit. A common objective in a device sizing problem is the total power consumed by the circuit. The optimization problem is to find the device sizes that satisfy the design requirements and are most power efficient [4].

There are a lot of methods to solve optimization problems. Simplex algorithm and Combinatorial algorithms can terminate in a determined number of steps. Iterative methods generate a sequence of improving approximate solutions until meeting a termination criteria. Iterative methods evaluate Hessians, gradients, or only function values to guide the optimization process. Those methods include Newton's method, Quasi-Newton methods, Gradient descent, Pattern search methods, and so on. Heuristic algorithms include those bio-inspired algorithms, e.g. Evolutionary algorithms, Artificial bee colony optimization, Particle swarm optimization, etc., and other state-of-the-art algorithms, e.g. Simulated annealing, Tabu search, etc.

1.2.1 Fitness Landscape

In optimization problems, a search space is defined as the space that contains all possible optimization variables of a problem. In another word, each optimization variable x is corresponding to a point in the search space. A fitness landscape is defined as the hyper-surface that represents the distribution of

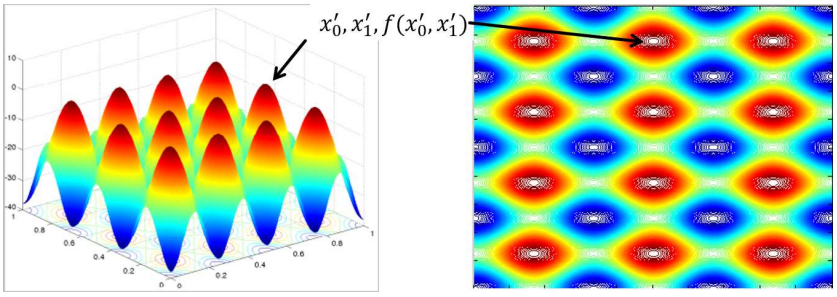


Figure 1.1: An example of fitness landscape

fitness values over all points in the search space: for any x, it is corresponding to a point $(x, f(x))$ on fitness landscape. For example, to maximize the function:

$$f(x_0, x_1) = -20 - 9(cos(6\pi x_0) + cos(8\pi x_1)) \quad x_0, x_1 \in (0, 1) \tag{1.2}$$

An optimization variable is $x = (x_0, x_1)$. The search space is a 2-dimension space ranging in (0,1) in each dimension. And the fitness landscape is a hyper-surface in a 3-dimension space with the coordinate $(x_0, x_1, f(x_0, x_1))$ as shown in the left of Fig. 1.1. Higher fitness is represented as red, and lower fitness is represented as blue. For the convenience of the following discussion, the the fitness landscape is also showed as contour line in the right of Fig. 1.1.

Solutions can be separated into two kinds: global solutions and local solutions. They are corresponding to global and local maximum points on the fitness landscape. All global and local maximum points of the fitness

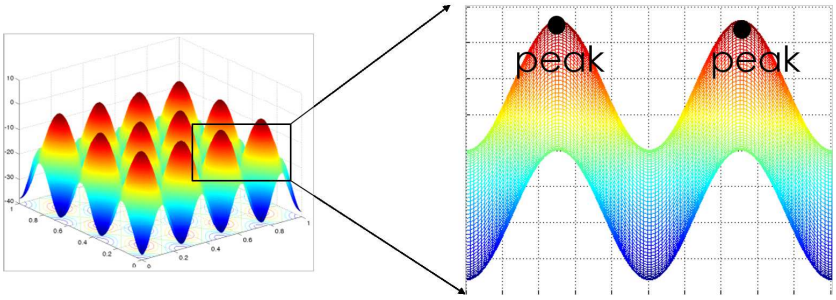


Figure 1.2: An example of peaks

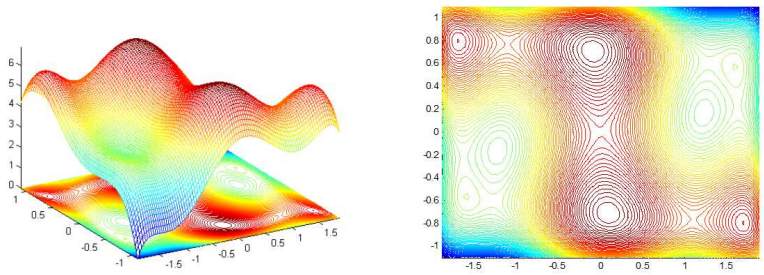


Figure 1.3: An example of global and local peaks

landscape are named as peaks, as shown in Fig. 1.2. In Fig. 1.1, there are 12 peaks, and all of them are global solutions. In Fig. 1.3, there are 2 global peaks and 4 local peaks.

1.3 Evolutionary Algorithm

Evolutionary Algorithms [2] are search heuristic that mimics the process of natural selection to solve optimization problems. It includes Genetic Algorithms (GA), Evolution Strategies (ES) and Genetic Programming (GP).

1.3.1 Genetic Algorithm

Genetic Algorithm (GA) [13, 3] is a heuristic algorithm to search approximate solutions on optimization problems. It is inspired by natural evolution process. In GA, the optimization variable of the problem, the vector x, is coded into a binary or a real-valued vector. A vector is called as an individual (or chromosome) which is corresponding to a point in the search space, and each dimension of the vector is named as a gene. A population P of GA consists of N individuals.

Genotype and phenotype are used to refer the representative of an individual in the GA, and the representative for calculating the fitness respectively. As mentioned above, genotype is a binary or a real-valued vector. Phenotype could be a tour in TSP, or an electronic circuit. In some tasks, genotype is the same as phenotype, e.g. in the example of maximizing Eq. 1.2. In other tasks, we need to map genotype to phenotype, e.g. TSP.

In the running of GA, the individuals should be randomly initialized before the beginning of optimization. GA optimizes the population by repeating a set of operation called generation. A generation usually consists of "Evaluation process", "Selection process", "Crossover process" and "Mutation process".

The Evaluation process calculates the fitness of all individuals by using

the fitness function. The Selection process has a role which decides survival individuals based on fitness in each generation. It selects N relative better individuals (bigger fitness) and collects them in a new set $P^{'}$. A common used selection operator is Tournament selection which randomly takes k individuals from the population and takes the best one as the selected one.

Crossover process works on the set $P^{'}$. It randomly takes two individuals from $P^{'}$ as parents and generates a random number. If the random number is smaller than the parameter crossover possibility P_c, then a new individual is created as child (offspring) by using a crossover operator and added to a set $P^{''}$. Otherwise the two parents are added to the set $P^{''}$. A common used crossover operator is Half Uniform Crossover which randomly copies genes to the child from the first or from the second parent. Crossover process finishes when $P^{''}$ is filled with N individuals.

Mutation process works on the set $P^{''}$. A common used mutation operator generates a random number at each gene, and if the random number is smaller than the parameter mutation possibility P_m, a new random number is generated and added to this gene.

Finally, the $P^{''}$ is set as the population of the next generation. Generations are repeated again and again until a termination condition is reached. A common used termination condition is a fixed number of generations or evaluations have been reached. After reaching the termination condition, the

best individual in the population is an approximate solution of the problem. The pseudo code of GA is shown in algorithm 1.1.

Algorithm 1.1 Genetic Algorithm

input:
N - number of individuals;
P_c - crossover possibility;
P_m - mutation possibility;
output:
P - Population;

Fill population P with N random individuals;
repeat
  Calculate the fitness of P;
  Apply Selection Operator;
  Apply Crossover Operator with P_c;
  Apply Mutation Operator with P_m;
until termination condition

1.3.2 CMA-ES

Evolutionary Strategy with Covariance Matrix Adaptation (CMA-ES) [20, 11] is a specific variant of evolution strategies (ES) that has been successful for treating correlations among object variables. It can adaptively adjust the mutation parameters of ES according to the searching history. Hence, CMA-ES does not require an experimental configuration. In contrast, in order to perform an optimization by using GA, we have to set some parameters preliminarily. Moreover, it is well-known that it has better optimization capability than GA. I shall provide here only a short description of the (1,λ)-

CMA-ES.

Given an initial search point $xmean^0$, it is updated in each generation in order to find a better solution. In generation g, first we create λ offspring x^g by mutating $xmean^g$. Second, the best offspring is chosen to become the search point of the next generation. The offspring are generated in this way:

$$x^g = xmean^g + \sigma \cdot B \cdot z, \tag{1.3}$$

The z is a vector of random variables sampled from the multivariate normal distribution. The σ is the global step size. The matrix B defines the mutation distribution. It is initialized as the *unity matrix*. The σ and B are adjusted according to the searching history.

To explain how to update σ and B, several new variables should be introduced. A matrix D is initialized as the *unity matrix*. A matrix C is initialized as:

$$C = B \cdot D \cdot transpose(B * D); \tag{1.4}$$

Two vectors pc, ps are initially filled with zero. I name the z that produces the best offspring as $zmeanw$.

The σ is updated in the way:

$$ps = cs \cdot ps + cs' \cdot (B \cdot zmeanw); \tag{1.5}$$

$$\sigma = \sigma \cdot exp((normalize(ps) - chiN)/chiN'); \tag{1.6}$$

Here, cs, cs', $chiN$ and $chiN'$ are parameters defined before the optimization.

The B is updated in the way:

$$pc = cc \cdot pc + cc' \cdot (B \cdot D \cdot zmeanw); \tag{1.7}$$

$$C = ccov * C + ccov' \cdot pc \cdot transpose(pc); \tag{1.8}$$

The new value of B and D are eigenvectors and eigenvalues of C. cc, cc', $ccov$ and $ccov'$ are parameters defined before the optimization.

1.3.3 Neuroevolution

Neuroevolution[10] refers to the method that uses evolutionary algorithms to train artificial neural networks (ANN) [21]. It is most commonly applied in artificial life, computer games, and evolutionary robotics.

ANNs are computational models inspired by nervous systems of animals (in particular the brain), and are used to estimate or approximate functions which map inputs to outputs. Figure 1.4 shows the architecture of ANN which has a three-layered feed-forward structure. This neural network is

expressed by the following equations at each time t.

$$h_j(t) = sigmoid\left(\sum_i w_{ij} y_i(t) + b_{j1}, T_{j1}\right) \tag{1.9}$$

$$z_k(t) = sigmoid\left(\sum_j u_{jk} h_j(t) + b_{k2}, T_{k2}\right) \tag{1.10}$$

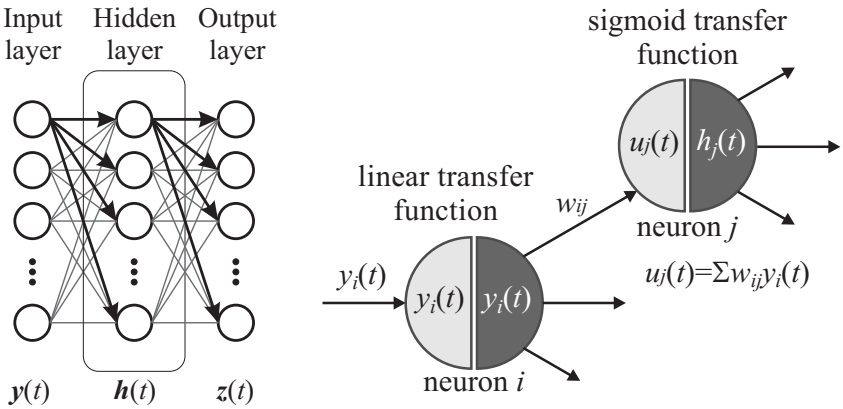
$$sigmoid(x, T) = \left(\frac{1}{1 + e^{-x/T}}\right) \tag{1.11}$$

where $y_i(t), h_j(t)$ and $z_k(t)$ represent input, hidden and output neurons, respectively. The parameters w_{ij} and u_{jk} denote the synaptic weights of ANN, b_{j1} and b_{k2} are bias values, and T_{j1} and T_{k2} are temperature coefficients of the sigmoid function $sigmoid(x, T)$.

Neuroevolution codes these parameters ($w_{ij}, u_{jk}, b_{j1}, b_{k2}, T_{j1}$ and T_{k2}) as a chromosome and applies evolutionary algorithms to optimize them. An example of the fitness of ANN is the Mean squared error(MSE) between the outputs of ANN and the desired outputs. In robotics, the fitness is the performance of the controller, which is an ANN, in a given task.

1.3.4 Particle Swarm Optimization

Particle Swarm Optimization (PSO) [17, 26] is inspired by the movement of organisms in a bird flock or fish school. PSO optimizes a problem by having



(a) Neuron Network (b) Neuron Architecture

Figure 1.4: Artificial neural network

a population of candidate solutions, here dubbed particles, and moving these particles around in the search space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position, but is also guided toward the best known positions in the search-space, which are updated as better positions found by other particles.

The position of a particle x_i is updated in this way:

$$v_i = w \cdot v_i + c1 \cdot r1 \cdot (p_i - x_i) + c2 \cdot r2 \cdot (pgd - x_i) \tag{1.12}$$

$$x_i = x_i + v_i \tag{1.13}$$

Here, w, $c1$, $c2$ are per-defined parameters. $r1$, $r2$ are two random numbers. p_i is the best point that x_i has ever found. pgd is the best point that all

particles have ever found.

1.3.5 Differential Evolution

Differential Evolution (DE)[6] optimizes a problem by maintaining a population of candidate solutions and creating new candidate solutions by combining existing ones according to a simple formulae.

To update a new candidate $x = (x_1, ..., x_n)$ in DE, a common method is to first create a random number r_i for each x_i. If $r_i \leq CR$, then x_i is calculated in this way:

$$x_i = a_i + F \cdot (b_i - c_i) \tag{1.14}$$

Otherwise, x_i is not changed. Here, CR (recombination factor), F (mutation factor) are two pre-defined parameters. a, b and c are three candidates randomly selected from the population.

1.3.6 Variable Mesh Optimization

Variable Mesh Optimization (VMO)[24] is a population-based algorithm in which the population are distributed as a mesh, and the offspring are generated by expansion and contraction processes.

1.4 Multimodal Optimization

Multimodal Optimization [33] is a major subfield of Mathematical Optimization. Multimodal optimization deals with tasks that involve finding all or at least some of the multiple solutions as opposed to a single best solution. The multiple solutions could all be global solutions or there could be a mix of global and local solutions. For example, in Fig. 1.1, Multimodal optimization aims to find all 12 peaks. However, original Mathematical Optimization only aims to find one of them. Multimodal Optimization is especially useful in engineering. For example, due to physical (and/or cost) constraints, the best results may not always be realizable. Evolutionary Algorithms, benefit by being population-based, has the potential ability to locate multiple solutions. However, this is against the natural tendency of EAs, which will always converge to the best solution, or a sub-optimal solution. Finding and maintenance of multiple solutions is wherein lies the challenge of using EAs for multimodal optimization.

1.5 The Existing Niching Techniques

Niching[25] is the technique that can help EAs to find and preserve multiple stable *niches*, or favorable parts of a fitness landscape possibly around multiple peaks, so as to prevent premature convergence to a single best solution.

A niching method should cooperate with a evolutionary algorithm to help the evolutionary algorithm to find multiple solutions. Their combination is named as a niching evolutionary algorithm.

In practice, a niche (subpopulation) is a group of individuals, and the goal of niching is to separate the entire population into several non-overlapping niches with the hope that different niches will converge to different peaks. (There are also some niching methods that do not explicitly create niches). Basically, there are two problem in using Niching technique:

1. How to design a niching method.

2. How to cooperate a niching method with a evolutionary algorithm.

In the following discussion, I catalogue niching methods by the way they work. By default, I assume a niching method can cooperate with GA or any evolutionary algorithms. Otherwise, I will explicitly indicate the evolutionary algorithm that a niching method can cooperate with.

1.5.1 Radius-based Niching Methods

The most famous kind of niching methods is the radius-based method. Radius-based methods are based on the assumption that peaks on the fitness landscape are kept a certain distance from each other. Hence, a parameter *radius* is introduced to indicate the desired distance between niches.

Fitness Sharing

Fitness sharing[25] modifies the fitness landscape by reducing the fitness in densely populated regions. It lowers each individual's fitness by an amount nearly equal to the number of similar individuals in the population. This method does not explicitly create niches.

The shared fitness *fs(i)* of an individual i is given by

$$fs(i) = \frac{f(i)}{\sum_{j \in P} Sh(i,j)} \tag{1.15}$$

where *f(i)* is the raw fitness, P is the population and Sh(i,j) is the sharing function. The most commonly adopted form of Sh is the following:

$$Sh(i,j) = \begin{cases} 1 - \frac{d(i,j)}{radius} & , \quad if \quad d(i,j) < radius \\ 0 & , \quad otherwise \end{cases} \tag{1.16}$$

where d(i,j) is the Euclidian distance between individuals i and j. The sharing function measures the similarity level between two individuals. It returns one if the individuals are identical, zero if their distance $d(i,j)$ is higher than a threshold of dissimilarity (radius), and an intermediate value at intermediate level of dissimilarity.

Clearing method

The clearing method [1] is very similar to fitness sharing but is based on the concept of limited resources of the environment. Instead of sharing the resources between all individuals of a single niche, clearing method assigns them only to the best members of the niche. In practice, individuals belong to the same niche if their distance in the search space is less than a dissimilarity threshold (radius). The capacity k of a niche specifies the maximum number of elements that this niche can accept. Thus, clearing preserves the fitness of the k best individuals of the niche and resets the fitness of the others that belong to the same niche.

Dynamic Fitness Sharing

Dynamic Fitness Sharing (DFS)[5] explicitly identify *seeds* (In the original paper, it is called "peak". But in this thesis, peaks are the property of fitness landscapes. So I use "seed" to instead of it.) and create niches from the population. A seed is defined as an individual that is more closer to a peak than any other individuals in the population. Seeds can be identified by Dynamic Seed Identification (DSI) as shown in algorithm 1.2. It selects q best individual which are at least r far from each other as seeds. Niches can be created by collecting individuals in the radius of the seeds. DFS runs

fitness sharing on each niche instead of in the entire population.

Algorithm 1.2 Dynamic Seed Identification (DSI)

```
input: Pop - array of population members;
       q - number of seeds to identify;
       r - niche radius;
output:S-a set of seeds

Sort Pop in decreasing fitness order;
NumSeeds=0;
S=∅;
for i=1..size of Pop do
  if Pop[i] is not within r of seeds in S then
    S=S ⋃ {Pop[i]};
    NumSeeds=NumSeeds+1;
  end if
  if NumSeeds==q then
    break;
  end if
end for
```

An example of the difference between standard genetic algorithm and DFS is given in Fig. 1.5 and Fig. 1.6. The standard genetic algorithm tends to select the two best individuals as parents as shown in the middle of Fig. 1.5. So the offspring created by standard genetic algorithm converge to a single peak as shown in the right of Fig. 1.5. On the contrary, DFS select seeds and creates niches as shown in the left of Fig. 1.6. Each niche selects parents and creates offspring independent, So DFS can converge to multiple peaks.

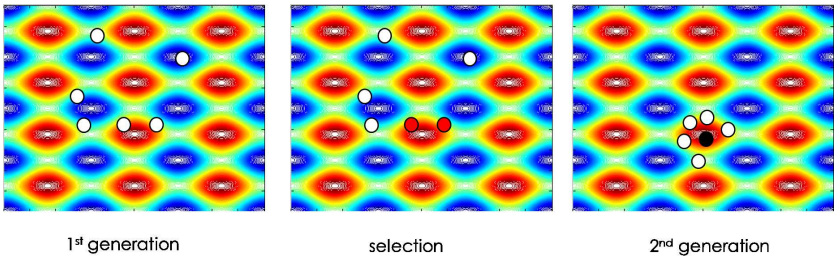


Figure 1.5: An example of standard genetic algorithm

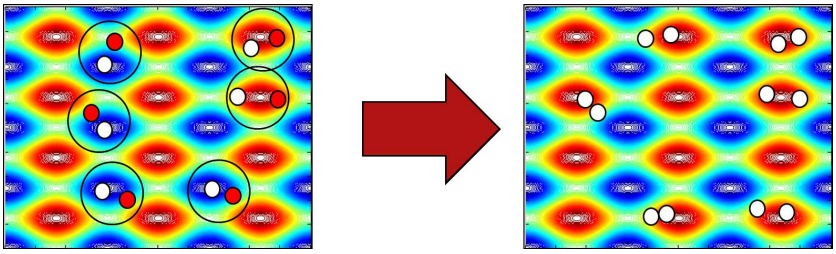


Figure 1.6: An example of dynamic fitness sharing

Dynamic Niching with CMA

Due to the nature of CMA-ES, it is hard to design a niching method for a single CMA-ES instance. Hence, niching CMA-ES methods are usually a hybridization of running multiple CMA-ES instances in parallel with a high-level control strategy. The control strategy dynamically adjusts the instances to avoid multiple instances searching in the same area.

The Dynamic Niching with CMA[27] is a well-known niching method for CMA-ES. It should be noted that the Dynamic Niching with CMA is in particular to the (1,λ)-CMA. A brief description of the algorithm follows.

The algorithm holds $q + 1$ search points. Each search point relates to a *CMA-sets*. A CMA-set includes the current search point, the covariance

matrix, the global step size and other parameters of a CME-ES instance. In each generation, first, each search point generates λ offspring based on its CMA-set. Second, the algorithm evaluates the fitness of $\lambda \cdot (q+1)$ offspring. Third, DSI is used to identify seeds from all offspring. Fourth, the identified seeds are chosen to become the new search points. Their CMA-sets are inherited from their parents and updated according to the CMA method. Fifth, if the number of identified seeds are fewer than q, the rest of the search points are randomly re-generated. Finally, the $(q+1)^{th}$ search point is randomly generated in order to explore more search space. This concludes a single generation loop, as given in algorithm 1.3.

Algorithm 1.3 Dynamic Niching with (1,λ)-CMA-ES: A Single Generation Loop

for all i=1..q+1 search points **do**
  Generate λ samples based on the CMA distribution of i;
end for
Evaluate fitness of the population;
Compute seeds of the $\lambda \cdot (q+1)$ individuals using the *DSI*;
for every seeds **do**
  Set seed as a search point of the next generation;
  Inherit the CMA-set and update it respectively;
end for
if N_s=number of seeds $< q$ **then**
  Generate $q - N_s$ new search points, reset CMA-sets;
end if
Reset the $(q+1)^{th}$ search point;

Niching CMA-ES with Adaptive Niche Radius

Ofer M. Shir [28] propose a Niching CMA-ES that can adaptively adjust the niche radius. The basic idea is to use the adaptive ability of CMA. Two approaches are considered. The first approach couples the radius to the step size mechanism, while the second approach employs the Mahalanobis distance metric with the covariance matrix mechanism for the distance calculation, for obtaining niches with more complex geometrical shapes.

1.5.2 Detect-Multimodal Method Based Niching Methods

The Multinational Genetic Algorithm (MGA)[15, 16] and Topological Species Conservation algorithm (TSC)[30, 29] use the detect-multimodal method (DMM) to replace the radius parameter of radius-based methods. The way that they selects seeds and creates niches inherits from DFS, except that they do not use radius to decide if two individuals are belong to the same niche. This is done by using DMM. In order to determine whether two individuals are belong to the same niche, DMM calculates the fitness of a number of *interior points* sampled on the line between two given individuals. If the fitness of all interior points are higher than the fitness of at least one of the two given individuals, then the two individuals are considered belong to the

smae niche.

MGA is the first DMM-based method. However, DMM spends many fitness evaluations on evaluating *interior points*, which can be considered as inefficient. TSC inherits DMM from MGA, but modifies other parts. It is proven to be more efficient than MGA.

1.5.3 Crowding Methods

Crowding methods insert new individuals in the population by replacing similar individuals. In DeJong's crowding [14], only a fraction of the global population specified by a percentage G (generation gap) reproduces and dies each generation. In this crowding scheme, an offspring replaces the most similar individual taken from a randomly drawn subpopulation of size CF (crowding factor) from the global population. In Mahfoud's deterministic crowding, DeJong's crowding is improved by introducing competition between children and parents [32]. After crossover and mutation, each child replaces the nearest parent if it has a higher fitness. Restricted Tournament Selection (RTS)[12] initially selects two individuals from the population to undergo crossover and mutation. After recombination, a random sample of CF individuals is taken from the population as in standard crowding. Each offspring competes with the closest sample individual. The winners are in-

serted in the population.

1.5.4 Other Niching Methods

Some newly proposed methods are hard to classified into the above catalogues. I describe some of them in this section.

Nearest Neighbor Differential Evolution Algorithm (DE/nrand)[9] is an improvement of the classic DE/rand algorithm for niching tasks. More specifically, in the mutation operation, the x_i in DE/rand is replaced by its nearest neighbor.

Dynamic Archive Niching Differential Evolution Algorithm (dADE/nrand)[8] is an improvement of DE/nrand. It employs an archive which uses a niche radius R. It also introduces an method to adaptively calculate the mutation factor F, recombination factor CR and the niche radius R.

Niching the CMA-ES via Nearest-Better Clustering (NEA2)[22] is a CMA-ES based niching method that uses Nearest-Better Clustering to create niches. Nearest-Better Clustering[23] assumes that the distance from a seed to the members in its niche is shorter than the distance between two seeds. Nearest-Better Clustering uses this property to separate the population into niches.

Niching Variable Mesh Optimization (NVMO)[19] is an improvement of VMO. It employs an archive, a local search method and a new combina-

tion operation which uses the best neighbour and the nearest current global optima to produce offspring.

1.6 The Problems or Challenges in the Existing Niching Techniques

Radius-based methods usually work fast, and are very successful in many complex tasks. However, it is hard to accurately set the radius parameter if the information of the fitness landscape is unknown which is very popular in real world tasks. Radius-based methods also assume fitness landscapes are even which can not be satisfied in many complex tasks. DMM-based methods can adapt for unknown landscapes. However it spends many fitness evaluations on evaluating interior points, which can be considered as inefficient. Hence, The purpose of the study is to propose a new niching method, which have the following new features:

1. The algorithm does not use the radius parameter.

2. The algorithm can create niches adaptively according to the fitness landscape.

3. The algorithm should overcome other niching methods on the ability of finding multiple solutions.

4. The algorithm should be efficient in computational time.

1.7 Conclusion

In this Chapter, I introduced our research background and objectives. Our main objective is to propose a niching evolutionary algorithm that can create niches adaptively according to the fitness landscape.

Chapter 2

Attraction Basin Sphere Estimation

2.1 Introduction

Radius-based niching methods [25, 1, 5, 27] usually work fast. However, it is hard to accurately set the radius parameter if the information of the fitness landscape is unknown which is very popular in real world tasks. Radius-based methods also assume fitness landscapes are even which can not be satisfied in many complex tasks. DMM-based methods [15, 16, 30, 29] can adapt for unknown landscapes. However it spends many fitness evaluations on evaluating interior points, which can be considered as inefficient.

This chapter focuses on proposing a niching method that is capable of

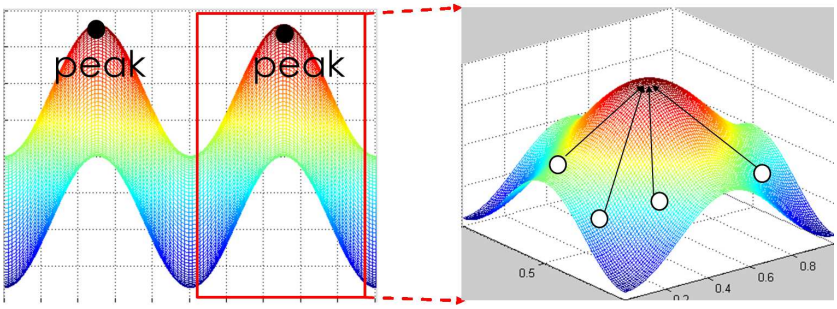


Figure 2.1: An example of attraction basin

adapting to different fitness landscapes in a reasonable time. The rest of this chapter is composed as follows. Section 2.2 analyses the property of fitness landscapes and describes the concept of Attraction Basin Sphere (ABS). Section 2.3 describes the Detect-Multimodal Method (DMM). Section 2.4 describes the Attraction Basin Sphere Estimation (ABSE). Section 2.5 gives some examples of using ABSE and discusses some features of ABSE. Section 2.6 concludes this chapter with some remarks. This chapter describes a fundamental algorithm used in all my papers [35, 36, 37, 34].

2.2 Attraction Basin Sphere

The attraction basin of a peak is defined as a region around the peak that satisfies:

1. For each points i in the region: the fitness of all points on the line from i to the peak are monotonically increasing.

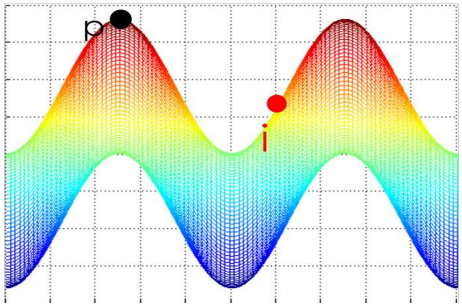


Figure 2.2: An example of multimodal

An example of the attraction basin is given in Fig. 2.1. The Attraction Basin is a very important concept for niching method. Because most optimization algorithms explicitly or implicitly use the gradients to find peaks, so if the fitness is monotonically increasing from point i to a peak p, we consider that an optimization algorithm can be guaranteed to find p beginning from i. Hence, from all points in an attraction basin, we consider that an optimization algorithm can be guaranteed to find the peak of this attraction basin. On the contrary, if the fitness is not monotonically increasing from the point i to a peak p, as in Fig. 2.2, we consider that an optimization algorithm can not be guaranteed to find p beginning from i.

Hence, attraction basins provide an ideal measurement for creating niches. All points in an attraction basin should be in the same niche. However, except some benchmark tests, we can not know attraction basins or peaks of a problem. To utilize attraction basins, I propose the concept of Attraction Basin Sphere (ABS).

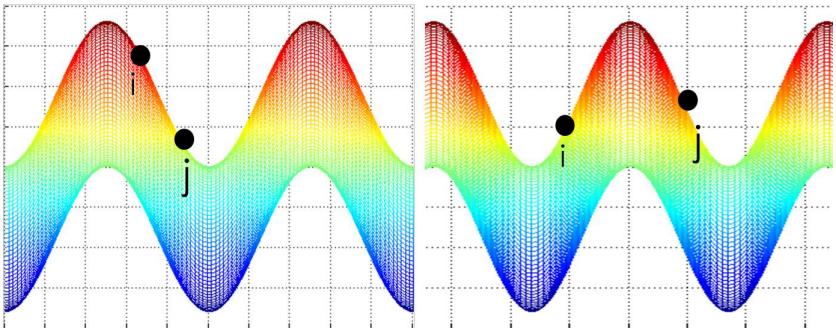


Figure 2.3: An example of unimodal

I generalize the above discussion. Two points i and j are unimodal is defined as:

1. all fitness values between point i and point j are better than at least one of i or j.

Two situations of unimodal are shown in Fig. 2.3. If two points are not unimodal, they are defined as multimodal, as shown in Fig. 2.2.

The ABS of a point p is defined as a hyper-sphere around p that satisfies:

1. p and each points in p are unimodal.

An example of ABS is given in Fig. 2.4. The basic idea of calculating the hyper-sphere is to find a radius that can exclude all multimodal points. This work is based on Detect-multimodal Method.

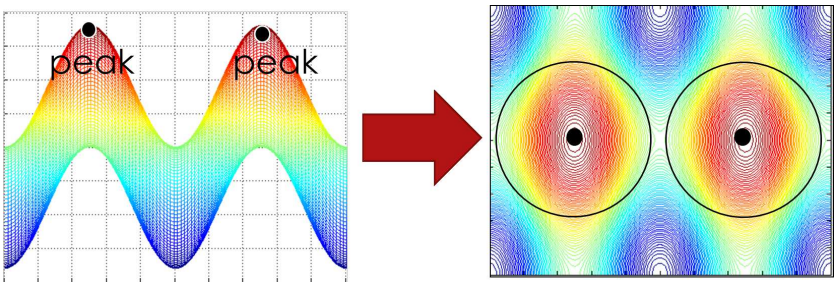


Figure 2.4: An example of ABS

2.3 Detect-multimodal Method

Detect-multimodal method (DMM)[15, 16] can analyse the *relationship* of two points on a fitness landscape. The term *relationship* here means, if the two points are unimodal or multimodal. A brief description of DMM follows. DMM takes two points A and B as arguments, and then generates a set of interior points by:

$$interior[j] = A + (B - A) \times gradation[j], \tag{2.1}$$

where $gradation[j]$ is the j'th entry of a user-defined set of gradations in the [0,1] interval. In case the fitness of all interior points is higher than at least one of A and B, it signals that A and B are unimodal, as shown in Fig.2.5, where green points are interior points. However, this result is not trustable, because the interval between interior points may not be small enough to keep the landscape monotonous. On the contrary, if the fitness of any interior points is lower than both A and B, it signals that A and B

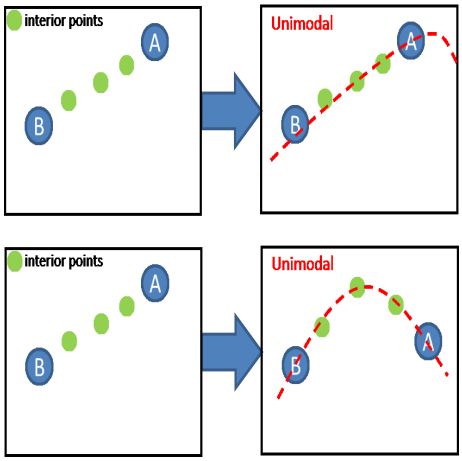


Figure 2.5: Two given points are detected unimodal

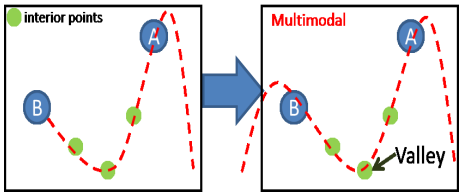


Figure 2.6: Two given points are detected multimodal

are multimodal, as shown in Fig.2.6. In this case, I name the lowest interior point as *valley*. This process is named as "*test* A with B".

2.4 The Estimation Method

According to the definition of *attraction basin sphere*, its radius can be estimated through valleys found by DMM. This method is named as Attraction Basin Sphere Estimation (ABSE).

A single test only detects the relationship on a single direction at a single distance. To find the ABS of a point i, we need to *test* i with several points

located at different distances and directions. Hence, the estimation needs to figure out two problems: First, how to calculate useful information from a single *test*; Second, how to combine the results of several *tests*.

Considering that we *test* a point A with B, according to the result of this *single test*, we can calculate two information: When A and B are multimodal, as Fig.2.6, the radius of *attraction basin sphere* of A should be smaller than the distance to the detected valley. Because the gaps between interior points, we can not exclude the possibility that a point which is lower than the currently found valley exists between A and B. Hence, I adjust the position of valley in proportion to the fitness of A, B and valley. The distance to the valley is calculated as Eq.2.2,

$$d_{valley} = \frac{D(A,B) \cdot (A.fitness - valley.fitness)}{A.fitness + B.fitness - 2 \cdot valley.fitness} \tag{2.2}$$

where D(A,B) is the distance between A and B. On the contrary, if A and B are unimodal, and A is better than B, as Fig.2.5, the radius of A should be bigger than the distance between A and B. The distance estimated from the first case and the second case are named as *upperbound* and *lowerbound* respectively.

When we *test* a point with several points, we obtain several *upperbound* and *lowerbound* values, which are used to calculate the radius of this point. I

only consider the smallest *upperbound* and the biggest *lowerbound*. They are named as *MinUpperbound* and *MaxLowerbound* respectively. The *MaxLowerbound* is used as the radius when it is smaller than the *MinUpperbound*. Otherwise, the *MinUpperbound* is used as the radius. The whole process is shown in algorithm 2.1, where in the relationship matrix M, each element m(i,j) records the *relationship* and the *valley* between point i and j. The *MaxLowerbound* and *MinUpperbound* are initially set to value 0 and ∞ respectively when p is identified as a seed. After that, each update is based on their current values.

2.5 Examples and Discussions

An example is given in Fig.2.7. The background of Fig.2.7 is the contour line of function:

$$f(x,y) = -((4 - 2.1x^2 + \frac{x^4}{3})x^2 + x \cdot y + (-4 + 4y^2)y^2) \qquad (2.3)$$

As in Fig. 1.1, higher fitness is represented as red, and lower fitness is represented as blue.

We want to estimate the radius of X. The relationship between point X and point *0*, point X and point *1* are multimodal. The relationship between

Algorithm 2.1 Attraction Basin Sphere Estimation (ABSE)

input: p - the point to be estimated;
T - array of points that has been *tested* with p;
M - relationship matrix;
output: updated p.radius;

if not initialized **then**
  p.$MaxLowerbound$=0;
  p.$MinUpperbound$=∞;
end if
$S_L = \{p.MaxLowerbound\}$;
$S_U = \{p.MinUpperbound\}$;
for all i=1..size of T **do**
  if M(p,T[i]).relationship==multimodal **then**
    Calculate d_{valley} using p,T[i],M(p,T[i]).$valley$ as arguments;
    $upperbound$=d_{valley};
    $S_U = S_U \bigcup upperbound$;
  else
    $lowerbound$ = distance between p and T[i];
    $S_L = S_L \bigcup lowerbound$;
  end if
end for
p.$MaxLowerbound$ = max(S_L);
p.$MinUpperbound$ = min(S_U);
p.radius = min(p.$MaxLowerbound$, p.$MinUpperbound$);

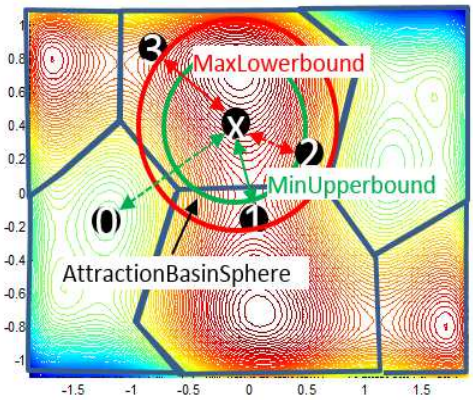


Figure 2.7: An example of attraction basin sphere estimation

point X and point *2*, point X and *3* are unimodal. Because the distance from X to point *1* (green full line) is smaller than the distance to point *0* (green dot line), so the distance to point *1* is the *MinUpperbound*. The *MaxLowerbound* is the distance to point *3* (red full line), because it is bigger than the distance to point *2* (red dot line). So the radius is the *MinUpperbound* (green circle), because it is smaller than the *MaxLowerbound* (red circle).

The fitness landscape may be irregular. So attraction basin may have irregular shapes. In this case, the attraction basin is filled by several ABSs, as shown in Fig. 2.8. The affect of this phenomenon will be described in the chapter 4.

As mentioned above, when two points are detected unimodal, this result is not trustable, because the interval between interior points may not be small enough to keep the landscape monotonous. Hence, the *MaxLowerbound* of

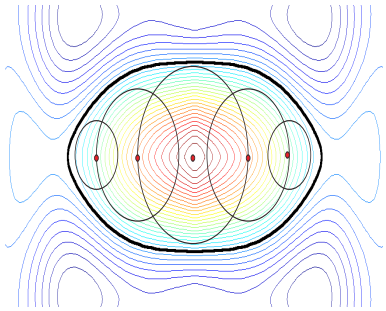


Figure 2.8: An example of an attraction basin covered by several ABS

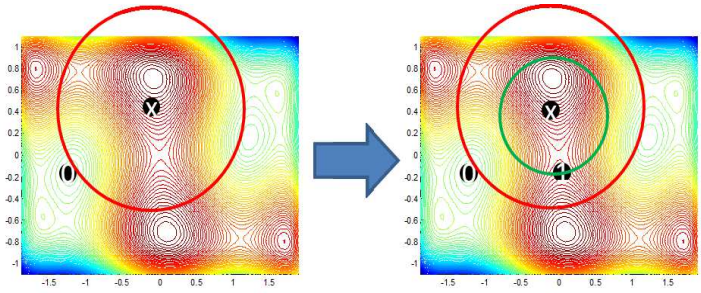


Figure 2.9: An example of how ABSE fixs mistakes

ABSE is not trustable. However, according to the mechanism of ABSE, a mistake in *MaxLowerbound* can be fixed by *MinUpperbound* if we make enough tests. An example is shown in Fig. 2.9. The point x is first tested with the point 1. They are multimodal. Unfortunately, ABSE makes a mistake and considers them as unimodal. Hence, *MaxLowerbound* is updated to the point 1 as the red circle in the left of Fig. 2.9. Next, the point x is tested with the point 2. This time ABSE correctly detects them as multimodal, so *MinUpperbound* is updated to the point 2 as the green circle in the right of Fig. 2.9. According to ABSE, the *MinUpperbound* is used as radius, so ABSE fixes the mistake and gets a correct radius.

In principle, the accuracy of ABSE depends on how many points are *tested* and how many interior points are used at a single *test*. However, in on our method, if a big number of points are *tested*, even a small number of interior points is adopted, the accuracy can also be guaranteed. Hence, we only consider that the accuracy depends on how many points are *tested*.

One of the inputs of ABSE is the array of points that has been *tested* with the point to be estimated. How to obtain the array is a problem in combining ABSE with evolutionary algorithms. The detail of choosing the array will be described in the following 3 chapters.

ABSE is different from the previous DMM-based method by the calculation of ABS. The previous methods use DMM to decide if an individual is belong to a niche. So every time we want to decide if an individual is belong to a niche, we need to run DMM between a seed and the individual. This mechanism makes the previous DMM-based methods inefficient, because DMM are run so many times. However, using ABSE, we can calculate ABS of a seed. When we want to decide if an individual is belong to a niche, we only need to check if the individual is in the ABS. In this mechanism, we only run DMM when estimating ABSs. So we decrease the times of running DMM, which makes ABSE-based methods more efficient than previous DMM-based methods.

2.6 Conclusion

In this section, I described a new niching method Attraction Basin Sphere Estimation. This method is capable of adapting to different fitness landscapes in a reasonable time. This is the fundamental algorithm of this study. It is combined with different EAs in the next 3 chapters.

Chapter 3

Attraction Basin Sphere Estimation Genetic Algorithm

3.1 Introduction

In the previous chapter, I describe a new niching method Attraction Basin Sphere Estimation. A niching method cooperates with an evolutionary algorithm to help the evolutionary algorithm to find multiple solutions. I combine ABSE and the genetic algorithm in this chapter. The combined algorithm is named as Attraction Basin Sphere Estimation Genetic Algorithm (ABSEGA). ABSEGA inherits some ideas from another DMM-based method Topological Species Conservation (TSC). I will discuss the features of the ABSEGA and compare it with TSC in this chapter.

The rest of this chapter is composed as follows. Section 3.2 introduces the Topological Species Conservation Algorithm. Section 3.3 describes the Attraction Basin Sphere Estimation Genetic Algorithm. Section 3.4 describes the experiments. Section 3.5 concludes this chapter with some remarks. This chapter is based on the paper [35, 36].

3.2 Topological Species Conservation Algorithm

Topological Species Conservation Algorithm (TSC) [30, 29] is a DMM-based niching genetic algorithm. Comparing with standard GA, TSC introduces three new steps. *Seed selection* step uses the DMM to identify seeds in the current population and separates the population into niches. It sorts individuals according to their fitness in decreasing order, and then tests every individuals in order with the identified seeds. If an individual is not unimodal with any seeds, then it is marked as a new seed. Otherwise, it is assigned to an existing niche. Each niche creates its own offspring, so the algorithm can find multiple solutions. *Seed conservation* step copies a seed back to the current population if the seed is multimodal with all individuals, or the seed is unimodal with some individuals, however, those individuals are worse than

the seed. *Integrate free individuals* step groups the new population according to the identified seeds by DMM. The algorithm is shown below, where *P(t)* is the population of the t generation.

Algorithm 3.1 TSC Algorithm

```
t = 0;
Initialize P(t);
Evaluate P(t);
while not termination condition do
  t = t + 1;
  Seed selection;
  Selection of P(t) from P(t − 1);
  Crossover and Mutation on P(t);
  Evaluate P(t);
  Seed conservation;
  Integrate free individuals;
end while
```

TSC is proven to be robust and adaptive for different fitness landscapes. However, it runs DMM too many times. Because each run of DMM needs to evaluate several interior points, the evaluations spent in one generation is in the range $[N, N + p_{interior}N^2]$, depending on the state of each generation and the parameter settings, where N is the population size and $p_{interior}$ is the number of interior points. Usually a big $p_{interior}$ is needed to guarantee accuracy. The results of interior points are discarded, which means these extra evaluations do not contribute to the convergence. This makes TSC computationally expensive.

3.3 Attraction Basin Sphere Estimation Genetic Algorithm

3.3.1 Overview

The motivation of this algorithm is if we can estimate the radius and use the estimated radius to create niches, we can get an adaptive and relatively fast algorithm.

The skeleton of ABSEGA is based on TSC. To improve the efficiency of TSC, I introduce a memory called Seed Archive, which is updated at each generation in the way:

1. Select seeds and insert them into Seed Archive.
2. Refine the ABSs of seeds in Seed Archive.
3. Delete covered seeds in the Seed Archive.

I create niches by assigning individuals to its nearest seeds in Seed Archive.

3.3.2 Seed Update

The Seed Archive is a memory which records the identified seeds and their relationship between each other. It consists of two parts:

1. PS - array of seeds.

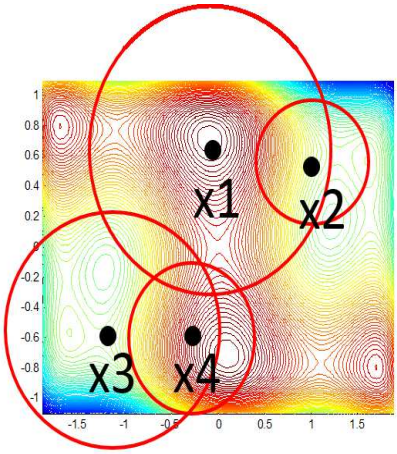


Figure 3.1: An example of cover

2. M - relationship matrix of seeds.

It is updated at each generation by three steps: seed selection, ABS refining and seed deletion.

Seed Selection

First, I define a seed A is covered by another seed B as:

1. B.fitness > A.fitness and distance(A, B) < B.radius.

2. B.fitness > A.fitness and distance(A, B) < A.radius.

An example is given in Fig. 3.1, where x2 is covered by x1, and x3 is covered by x4.

At each generation, I find up to N_{expand} seeds and insert them into Seed Archive. To find a seed, I first select an individual from the population through Tournament Selection. Second, if this individual is not covered by

any existing seeds, I consider it as a seed and insert it into Seed Archive. This process is shown in algorithm 3.2.

Algorithm 3.2 Seed Selection

input: P - array of population members;
N_{expand} - number of seeds to identify;
SA - Seeds Archive;
output: updated $SAi.PS$

$k = 0$;
while $k < N_{expand}$ **do**
  select an individual x from P through Tournament Selection;
  $isoverlap = FALSE$;
  for each seed i in SA **do**
    if x is covered by i **then**
      $isoverlap = TRUE$;
      break;
    end if
  end for
  if $isoverlap$ is FALSE **then**
    insert x into $SA.PS$;
    $k = k + 1$;
  end if
end while

ABS Refining

In order to use ABSE to estimate the ABS of a seed, I need to provide some points that are tested with the seed. Given a seed x, there are two ways to select the partner point y for making the test. The first way is called *mutual test*. The seed is tested with another seed near it. I test the seed x with other seeds in order starting from the closest one, up to the maximum of $p_{neighbour}$

neighbors. This process is shown below in algorithm 3.3.

Algorithm 3.3 Mutual Test

input: x - the seed to be estimated;
$p_{neighbour}$ - maximum number of neighbors;
SA - Seeds Archive;
output:updated $SA.M$

for i=1 to $p_{neighbour}$ **do**
  y = the i-th neighbor of x in SA;
  if y has not been tested with x **then**
    test y with x and record their relationship in $SA.M$;
    break;
  end if
end for

The second way to choose the point y is via *internal test*. A random point is generated within ABS of the seed x, with the intention of testing whether ABS still contains multimodal points. The process is shown below in algorithm 3.4. Because y is not a seed, so I records y and its relationship in another memory SA'.

Algorithm 3.4 Internal Test

input: x - the seed to be estimated;
SA - Seeds Archive;
output:SA';

$SA' = \emptyset$
y = a random point generated in ABS of x;
insert y into $SA'.PS$;
test y with x and record their relationship in $SA'.M$;

The algorithm of refining ABS is shown in algorithm 3.5. The algorithm runs N_{mutual} mutual tests and $N_{internal}$ internal tests. In mutual test, $p_{interior}$·

N_{mutual} interior points are evaluated. $p_{interior}$ is the number of interior points in one test. However, because the randomly generated point y also needs to be evaluated, $(p_{interior}+1)\cdot N_{internal}$ points are evaluated in internal test. The y and its relationship with seeds are stored in SA', so I update ABSs of seeds in SA using ABSE with both SA and SA' as arguments. The y in internal test and all interior points evaluated in both mutual test and internal test are stored in a set S, which will be used to accelerate the evolutionary process as described later.

Algorithm 3.5 ABS Refining

input: SA - Seeds Archive;
output:updated $SA.M$
S - all points evaluated in tests;

Update $SA.M$ by select N_{mutual} seeds and apply Mutual Test;
Get SA' by select $N_{internal}$ seeds and apply Internal Test;
Update ABSs with both SA and SA' as arguments;
$S = SA'.PS \bigcup$ all interior points.

Seed Deletion

The size of Seed Archive is limited by a parameter N_{memory}. If the size exceeds N_{memory}, the worst seeds in Seed Archive are deleted until the size is equal to N_{memory}. After the update of ABS, some seeds, which are not covered by other seeds before the update, will become covered by other seeds, those seeds also need to be deleted. This method is shown in algorithm 3.6. In

ABSEGA, N_{memory} is set bigger than the number of peaks to find. This strategy can decrease the possibility of deleting a seed that is low in fitness, but close to a undiscovered peak.

Algorithm 3.6 Seed Deletion

```
input: SA - Seeds Archive;
output:updated SA;

for each seed i in SA do
  for each j in SA, i ≠ j do
    if i is covered by j then
      delete i;
      break;
    end if
  end for
end for
if size(SA) > N_memory then
  delete the worst seeds until the size of SA = N_memory;
end if
```

3.3.3 Seed Conservation

Seed Conservation is the method that copies a seed back to the population when the population loses the ability to further improve this seed. The number of seeds maintained by TSC is usually set to 20% of the population size. So TSC can conserve all identified seeds in the population from generation to generation. However, as described above, ABSEGA aims to maintain much more seeds. So ABSEGA uses a different seed conservation strategy in which only up to $N_{conserve}$ seeds can be copied to the population at each generation.

The method is shown in algorithm 3.7. The method first randomly selects a seed i. Second, the method tries to find an individual j that in the ABS of i. If there are multiple individuals in the ABS of i, I take the worst one as j. If there is no such individual, I take the worst *unprocessed* individual as j and mark j as *processed*. Third, if the fitness of i is bigger than the fitness of j, I copy i to j.

Algorithm 3.7 Seed Conservation

input: P - array of population members;
SA - Seed Archive;
output:updated P;

$k = 0$;
while $k < N_{conserve}$ **do**
  random select a seed i from SA;
  find the worst individual j from P, j in the ABS of i;
  if j do not exist **then**
    find the worst unprocessed individual j from P;
    mark j as processed;
  end if
  if $i.fitness > j.fitness$ **then**
    $j = i$;
  end if
  $k = k + 1$;
end while

3.3.4 ABSEGA Algorithm

Here, the entire algorithm is presented. Seed Archive SA is initialized to empty. At generation t, ABSEGA first selects seeds from population $P(t-1)$. Second, ABSs are updated and all points evaluated in this step are stored

in a set S. Next, I delete covered seeds and run seed conservation. After that, I create P' by combine the population $P(t-1)$ and the set S. The P' is divided into niches by assigning each member of P' to its nearest seed. Finally, ABSEGA generates the population $P(t)$ as DFS. This process in shown in algorithm 3.8.

The evaluations spent in one generation is $N+p_{interior}\cdot(N_{mutual}+2N_{internal})$, where N is the population size. I use N_{mutual} and $N_{internal}$ much smaller than N. And $p_{interior}$ is set to smaller than that used in TSC. Hence, ABSEGA spends fewer evaluations in one generation than TSC. Furthermore, the evaluations spent in updating ABS contribute to build the next generation through the reproduction on P'. This feature makes ABSEGA more efficiently.

Algorithm 3.8 ABSEGA

$t = 0$
SA=$\emptyset$; - Seed Archive
Initialize $P(t)$;
Evaluate $P(t)$;
while not termination condition **do**
  $t = t + 1$;
  Seed selection to SA;
  ABS Refining on SA and store evaluated points in S;
  Seed Deletion on SA;
  Seed Conservation to $P(t-1)$;
  $P' = P(t-1) \cup S$;
  Reproduce $P(t)$ through P';
  Evaluate $P(t)$;
end while

3.4 Experiments

The experiments are performed on 7 benchmark test functions. The decision variables are coded in real numbers. The ability to detect multiple peaks, the ability to escape from local peaks and the computational cost are examined and compared with TSC. Finally, the effect of the parameter settings on the results is discussed.

3.4.1 Test Function

I select 7 representative test functions for our experiments from Stoean's paper[29].

1. Introduce F1: Waves Function

$$F1(x,y) = (0.3x)^3 - (y^2 - 4.5y^2)xy - \\ 4.7\cos(3x - y^2(2+x))\sin(2.5\pi x) \tag{3.1}$$

In the interval $x \in [0.9, 1.2]$, $y \in [-1.2, 1.2]$, this function has an irregular shape. It has 1 global optimum and 9 local ones, 4 of them being located on the boundary.

2. Introduce F2: Six-Hump Camel Back Function

$$F2(x,y) = -((4 - 2.1x^2 + x^4/3)x^2 + xy + (-4 + 4y^2)y^2) \tag{3.2}$$

In the interval $x \in [-1.9, 1.9]$, $y \in [-1.1, 1.1]$, this function is symmetrical about the origin. It has 2 global peaks and 4 local ones. Two of local peaks are considerably lower than other peaks.

3. Introduce F3: Shubert Function

$$F3(x, y) = \sum_{i=1}^{5} i \cos[(i+1)x + i] \cdot \sum_{i=1}^{5} i \cos[(i+1)y + i] \tag{3.3}$$

In the interval $x, y \in [-10, 10]$, this function has 18 peaks located regularly on a bumpy surface. The attraction basins of those 18 peaks occupy very small area in the landscape.

4. Introduce F4: Ursem F3 Function

$$\begin{aligned} F4(x, y) = \sin(2.2\pi x + 0.5\pi) \cdot \frac{2 - |y|}{2} \cdot \frac{3 - |x|}{2} + \\ \sin(0.5\pi y^2 + 0.5\pi) \cdot \frac{2 - |y|}{2} \cdot \frac{2 - |x|}{2} \end{aligned} \tag{3.4}$$

In the interval $x \in [-2.5, 3]$, $y \in [-2, 2]$, this function has 1 global optimum and 4 local ones on a smooth landscape. Peaks have regular ellipse shapes.

5. Introduce F5: Shifted Rastrigin Function

$$F5(\vec{x}) = \sum_{i=1}^{D} (x_i^2 - 10 \cos(2\pi x_i) + 10) \tag{3.5}$$

In the interval $x_i \in [-5, 5]$, this function has a spiny surface. It has a global

optimum at the origin surrounded with a large number of local peaks which only has a small difference on height from the global one.

6. Introduce F6: Rotated Hybrid Composition F21 Function

The detail of this function can be found in Suganthan's paper on CEC2005 [31]. It is composed of Ackley, Rastrigin, Sphere, Weierstrass, and Griewank functions. Besides the huge number of local peaks, its attraction basin is very small. It is extremely difficult to find the global optimum of this function in high dimension.

7. Introduce F7: Keane's Bump problem

$$F7(\vec{x}) = |\sum_{i=1}^{D}\cos^4(x_i) - 2\prod_{i=1}^{D}\cos^2(x_i)|/\sqrt{\sum_{i=1}^{D} ix_i^2}$$
$$subject\ to\ \prod_{i=1}^{D} xi > 0.75, \sum_{i=1}^{D} xi < \frac{15D}{2} \tag{3.6}$$

The interval of F7 is $x_i \in [-10, 10]$. It is a constraint problem and has lots of local peaks. The constraints in Keane's Bump problem are treated by simply decreasing the fitness of infeasible individuals to zero.

3.4.2 Performance Criteria

This paper employs 3 performance criteria from TSC.

The peak ratio (PR) is the fraction between the number of found peaks

and the number of total peaks to be found, as shown in Eq. 3.7, where NPF_i denotes the number of found peaks calculated in the end of the i-th run. NKP is the number of global peaks. NR is number of runs. A peak is considered to be found if there is a seed in Seed Archive that is close to a peak with 10% error margin.

$$PR = \frac{\sum_{i=1}^{NR} NPF_i}{NKP \cdot NR} \tag{3.7}$$

The peak accuracy (PA) measures the difference between the fitness values on peaks and seeds. For each peak to be found, the absolute fitness difference to the nearest seed in Seed Archive is calculated. Then, all these differences are summed together as the value of peak accuracy.

The distance accuracy (DA) tries to measure the dissimilarity between the peaks and seeds. It is calculated in the same way as peak accuracy, except that the fitness difference is replaced by the Euclidean distance between peaks and seeds.

The nearest solution is replaced by the fittest solution for the calculation of peak accuracy and distance accuracy in F6 with 10 dimensions, because original peak accuracy and distance accuracy criteria may not count the fittest solution. The reason is both ABSEGA and TSC fail to locate attraction basin of the global peak in this test. When applying those criteria to

TSC, the seeds in the Seed Archive are replaced by the individuals in the population.

3.4.3 Performance Comparison

The experiments for performance comparison are separated into two groups. The first group aims to test the ability of detecting multiple peaks. The algorithms are tested on functions F1 to F4 in this group. All functions are two-dimensional. The numbers of peaks counted by the criteria for F1 to F4 are 10, 6, 18 and 5 respectively.

The second group aims to test the ability to escape the local peaks (local optima) and find the global peak (global optima). The tests include F5 with 10 dimensions, F6 with 2 and 10 dimensions, F7 with 20 dimensions. Only the global peak is considered in the criteria.

All parameters are manually tuned in the experiments. The population size and crossover rate are set to 100 and 0.6. The size of tournament selection is 3. The algorithm stops when the maximum generation is reached. It is 3000 for F6 with 10 dimensions and 1000 for other functions.The mutation rate p_m are set depending on the test functions, as shown in Table 3.1. F1, F3 and F4 share the same mutation rate. However, I use a very high mutation rate in F2, because two peaks of F2 are much lower than others. The range

of each dimension in genotype is [-10,10], it is mapped to the range of test function in the phenotype.

For ABSEGA, the N_{memory}, $N_{internal}$, $N_{conserve}$, $p_{neighbour}$ are set to 100, 10, 10 and 2 respectively for all test functions. The number of interior points $p_{interior}$ is set to 2 for F7, and 1 for others. The N_{expand} is set to 3 for all functions. N_{mutual} is set to the value of $p_{neighbour} \cdot N_{expand}$. For TSC, according to Stoean[29], the number of interior points of detect-multimodal is set to 4. The number of seeds is 20% of the population size. I perform 10 runs for each test function and outline the average results in Table 3.2. I implement TSC for the experiments.

For the first group, there are only slight differences between TSC and ABSEGA on the results of F1, F3 and F4. But ABSEGA is significant difference on F2. The peak ratio of both algorithms exceed 0.9 on those 3 functions, which means both algorithms succeed on detecting almost all peaks.

For the second group, TSC performs slightly better than ABSEGA on F5 and F7. However, the peak ratio of ABSEGA is 0.8 on F7, which is still a acceptable result. On F6 with two dimensions, ABSEGA performs significantly better than TSC. On F6 with 10 dimensions, neither algorithm is able to detect the global peak. Comparing the peak accuracy, ABSEGA is better than TSC which means the fitness obtained by ABSEGA is higher

Table 3.1: Mutation rate

Parameters	F1	F2	F3	F4	F5	F6 2D	F6 10D	F7
p_m ABSEGA	0.1	0.8	0.1	0.1	0.1	0.1	0.6	0.1
p_m TSC	0.1	0.8	0.1	0.1	0.1	0.8	0.1	0.1

Table 3.2: Criteria value of ABSEGA and TEC

Tests	PR		PA		DA	
	ABSEGA	TSC	ABSEGA	TSC	ABSEGA	TSC
F1	0.96	0.97	0.772	0.292	1.363	0.258
F2	0.92	0.68	0.416	0.637	0.408	0.743
F3	1.00	0.97	0.053	66.59	0.016	0.479
F4	0.95	1.00	0.163	0.011	0.187	0.025
F5	0.91	1.00	0.158	0.181	0.026	0.028
F6 2D	0.72	0.17	206	254	0.392	0.081
F6 10D	0.00	0.00	451	562	12.56	12.85
F7	0.80	1.00	0.106	0.124	4.137	0.904

than TSC. Both methods have slightly difference on distance accuracy.

In conclusion, the ability of ABSEGA to detect multiple peaks on various types of landscapes is demonstrated through the results of F1-F4. Furthermore, the ability to escape from local peaks on regular, highly irregular and constraint landscapes is demonstrated through F5-F7. Both algorithms perform slightly better than the other in different aspects, so they can be considered as having similar ability to solve the test functions.

3.4.4 Computational Cost

A major advantage of ABSEGA is that it is considerably more efficient than TSC in terms of computational cost.

In order to compare the computational cost of the two algorithms, I

consider the peak ratio for F1-F4, and the peak accuracy for F5-F7 as criteria. The parameters are set according to the previous experiments. The direct comparison on the number of evaluations of two algorithms may be unfair, because the number of interior points in most cases is set to 1 for ABSEGA but 4 for TSC. So I divide the results by the number of interior points. Because utilizing more interior points leads to a more accurate result, this average result can be considered to provide a conservative estimation.

For each test function, I separate the range of the criterion averagely into 100 levels and calculate time ratio (TR) which is the fractions between the evaluations spent by TSC and ABSEGA to reach the same level, as shown in Eq. 3.8.

$$TR = \frac{evaluations\ of\ TSC}{evaluations\ of\ ABSEGA} \tag{3.8}$$

The range of the criterions considered is 0 to 1 for F1-F4, 125 to 0 for F5,600 to 0 for F6 with 2 dimensions, 1500 to 0 for F6 with 10 dimensions, 0.65 to 0 for F7. As shown in Fig. 3.2, the x-axis is TR with log 10-transformed, and the y-axis is the levels.

I define the low, middle and high level of the criteria. For F1-F4, they are set to 40%, 60% and 80% for peak ratio. For peak accuracy of the F5-F7, they are set to 16, 8 and 4 for F5, 840, 540 and 440 for F6, and 0.48, 0.32 and 0.16 for F7. Table 3.3 shows the evaluations needed by the two algorithms

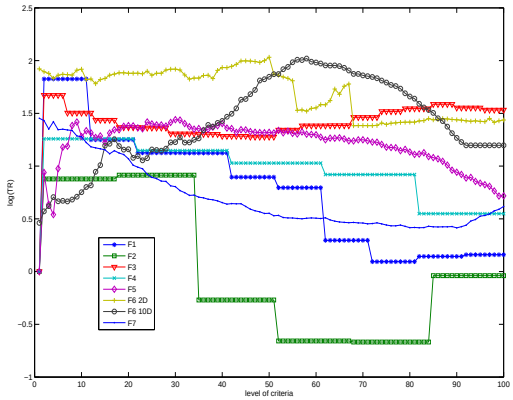


Figure 3.2: TR of TSC and ABSEGA to reach the same level

Table 3.3: Computational cost to reach three levels

Tests	low level		middle level		high level	
	ABSEGA	TSC	ABSEGA	TSC	ABSEGA	TSC
F1	580	3857	2010	8115	4160	18190
F2	700	3270	1933	6672	11433	9453
F3	6140	86625	8900	127850	13650	253825
F4	480	6973	1040	9132	1730	10915
F5	9940	61740	22150	111392	42050	183357
F6 2D	18	600	202	5360	633	12496
F6 10D	31050	440097	61550	721625	178350	775837
F7	1765	34967	7475	78930	19600	242065

to reach the three levels.

Through Fig. 3.2 and Table 3.3, it can be clearly seen that TSC spends much more evaluations to reach the same level than ABSEGA. On average, TSC spends 11 times evaluations, with a standard deviation of 9.4, to reach the high level criteria as ABSEGA. Considering the presented results are conservative results, ABSEGA can be considered as performing significantly more efficiently than TSC.

Table 3.4: Discrete points for sampling

Function	N_{memory}	N_{expand}	$N_{internal}$	$N_{conserve}$	$p_{neighbour}$
F1-F6	30, 60, 90	3, 6, 9	5, 10, 15	10, 20, 30	2, 4, 6
F7	30, 60, 90	1, 2, 4	2, 5, 7	10, 20, 30	2, 4, 6

3.4.5 Internal analysis of ABSEGA

I randomly sample 50 settings for parameters introduced by ABSEGA and test them on F1-F7. Each parameter is sampled from a discrete set, as shown in Table 3.4. Although each set has only 3 elements, it covers the range of common settings of each parameter. Other parameters are set according to the previous experiments. Each setting is performed 10 times. I use the peak ratio as criterion for F1-F4, and the peak accuracy for other functions.

I group the result of 50 settings by the sampling points and show the box plots for each test function in Fig. 3.3-3.6. The mean, standard deviation number of criterion values are shown in Table 3.5.

On F6, the standard deviation is high which means different settings have different performances. According to Fig. 3.5 and 3.6, N_{memory} impacts the performance on both F6 with 2 and 10 dimensions. N_{expand} and $N_{internal}$ impact the performance on F6 with 10 dimensions. However, the standard deviation and the difference between the performance of sampling points is low on F1-F5 and F7. It means on most functions, the randomly sampled settings have nearly equal performances .

Table 3.5: Mean and standard deviation of result of randomly sampled settings

results	F1	F2	F3	F4	F5	F6 2D	F6 10D	F7
mean	0.95	0.91	0.98	0.94	0.023	159	474	0.124
std	0.07	0.13	0.02	0.06	0.044	132	65	0.03

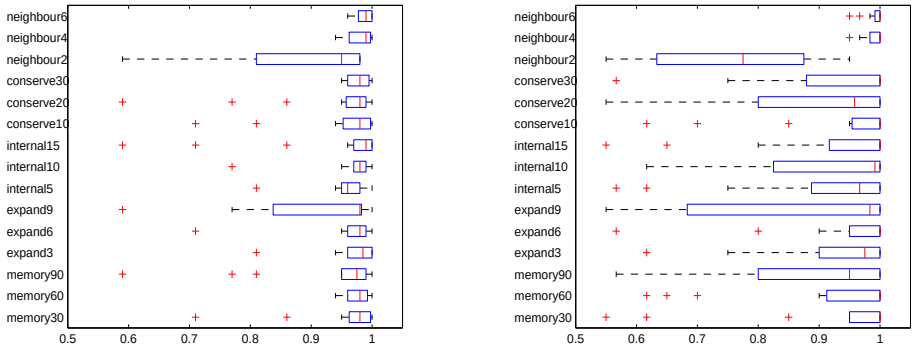


Figure 3.3: Box plot of result grouped by sampling points on F1(left), F2(right)

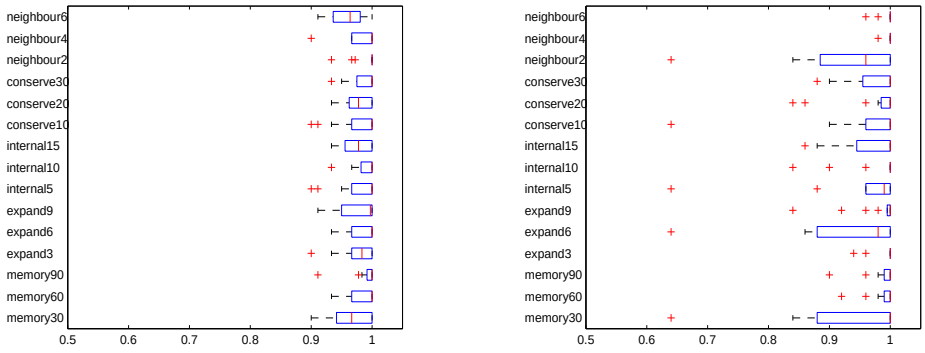


Figure 3.4: Box plot of result grouped by sampling points on F3(left), F4(right)

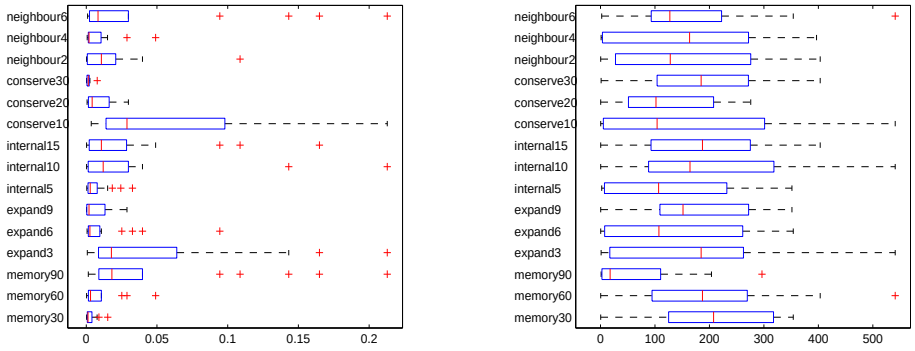


Figure 3.5: Box plot of result grouped by sampling points on F5(left), F6 2D(right)

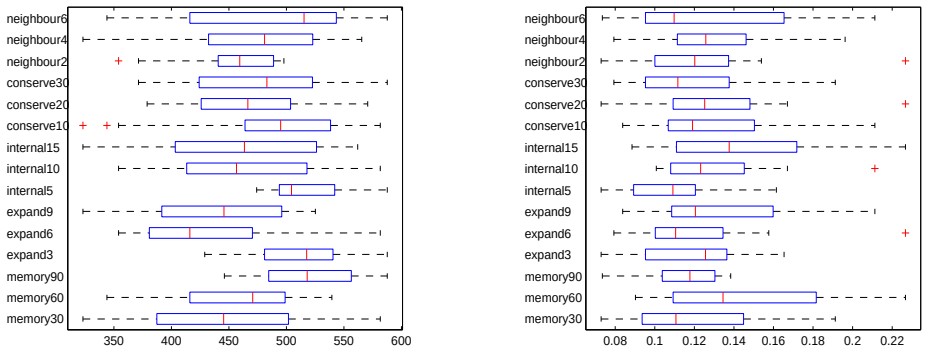


Figure 3.6: Box plot of result grouped by sampling points on F6 10D(left), F7(right)

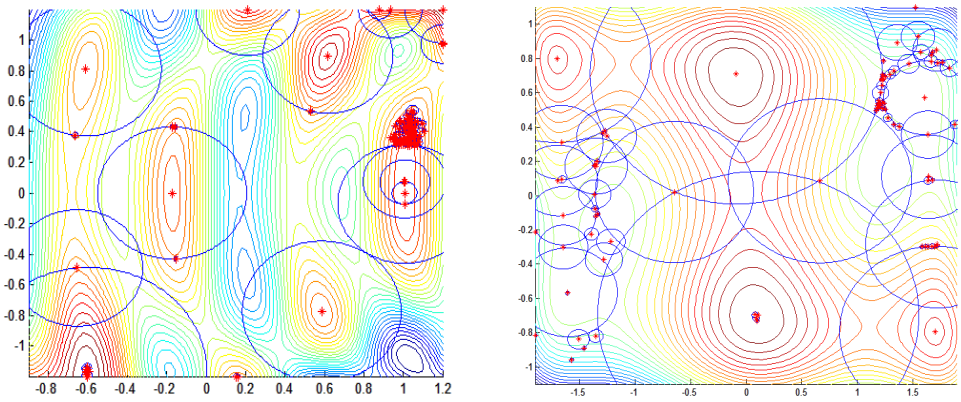


Figure 3.7: F1(left) and F2(right) results visualized

3.4.6 Visualization

The contour line of first test function group and the results obtained are visualized in Fig. 3.7-3.8. The points represent the seeds, and the circles around them represent the estimated ABS. As we can see, all peaks are found and their ABSs are properly estimated. The seeds and ABSs cover most parts of attraction basins on landscape and no ABS covers several peaks. Hence, ABSEGA can be considered successful on estimating the ABS. The only thing that looks strange is that some seeds are so close to each other. The reason is the size of Seed Archive N_{memory} is bigger than the number of peaks on the fitness landscape, so some seeds can not locate a peak, resulting in crowding together.

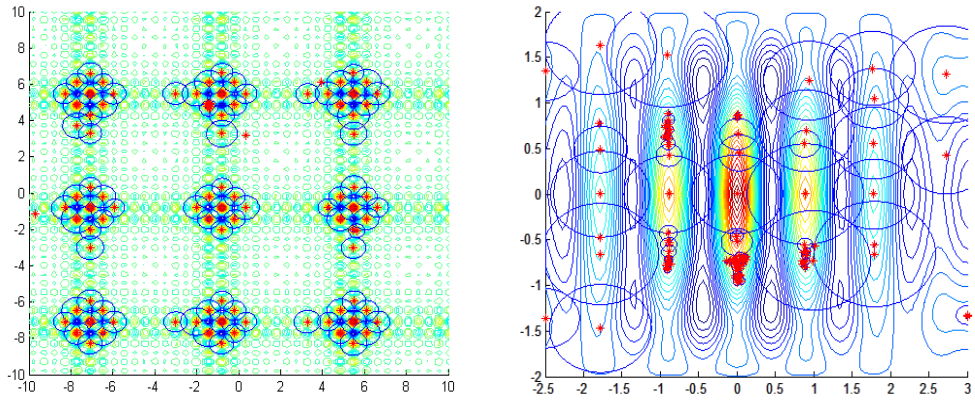


Figure 3.8: F3(left) and F4(right) results visualized

3.5 Conclusion

I introduced an adaptive and relatively fast niching genetic algorithm - ABSEGA. The experimental result showed that ABSEGA has ability to detect multiple peaks and escape from local peaks. The results indicated that ABSEGA has a similar ability to solve multimodal optimization problems as TSC. Considering the computational cost, ABSEGA was found to perform significantly more efficiently than TSC. Examining the randomly sampled parameter settings, the results were found to be insensitive to different parameter settings on most test functions.

Chapter 4

Attraction Basin Sphere Estimation Genetic Algorithm for Neuroevolution

4.1 Introduction

Attraction Basin Sphere Estimating Genetic Algorithm (ABSEGA) is a combination of ABSE and the genetic algorithm. It is adaptive to fitness landscapes and relatively fast.

ABSEGA works well on benchmark test problems, which only have some global peaks and a few number of local peaks. However, it does not work well on some high dimensional space problems (see the results of f6 10d

in ABSEGA). Because those problems usually have a lot of local peaks, and some local peaks may be on the surface of a global peak. So even too points are very close to each other, it is still possible for them to be multimodal. This is against the assumption of ABSEGA, which assumes the fitness landscape can be covered by several large ABSs. All these factors decrease the performance of ABSEGA in high dimensional problems.

Neuroevolution is a form of machine learning that uses evolutionary algorithms to train Artificial Neural Networks (ANNs). In standard Neuroevolution, a fitness function should be carefully designed in order to obtain a desired result. However, with the power of niching methods, a not well-designed fitness function can also lead to a desired result. Because niching methods can find multiple optima, and this ability increases the possibility of finding the desired result. However, Neuroevolution problems are usually high dimensional which is not suitable for ABSEGA.

In this chapter, ABSEGA is improved for Neuroevolution problems. I calculate a new variable *importance* to determine if a seed is more likely to be a global solution. So the algorithm can track global peaks among even hundreds of local peaks. This ability guarantees the algorithm can deal with high dimensional space. The new algorithm is named as ABSEGA2. I examine our method in benchmark tests and a robotic arm problem, in which I evolve an ANN to control the arm to catch balls.

The rest of this chapter is composed as follows: Section 4.2 explains the problems of ABSEGA in high dimensional space. Section 4.3 describes the seed importance calculation (SIC). Section 4.4 describes ABSEGA2. Section 4.5 describes the experiments. Section 4.6 concludes this study with some remarks. This chapter is based on the paper [37, 34].

4.2 Problems of ABSEGA in High Dimensional Space

There are usually much more local peaks in high dimensional space than low dimensional space, as shown in Fig.4.1. In low dimensional space Fig.4.1.a, there are only two peaks exist. However, in high dimensional space Fig.4.1.b, there are a lot of peaks. Two peaks marked by triangles are considered *important*, because they are the highest peaks in their own area. The peaks marked by circles are considered *unimportant*, because they are on the surface of the two important peaks.

ABSEGA uses a very simple way to differ important and unimportant peaks. When there are more than N_{memory} seeds in Seed Archive, ABSEGA simply deletes the worst one. However, this method may cause a problem. As in Fig.4.1.b, the important peak on the left is worse than the important

peak on the right, and one unimportant peak on the right. So if N_{memory} is set to 2, the important peak on the left will be kept out of Seed Archive. However, as the meaning of niching, 2 important peaks should be kept in the Seed Archive. Hence, ABSEGA may converge to an important peak with a lot of unimportant peaks around it, which leads to a low diversity in the population. In chapter 2, I mentioned that when an attraction basin is irregular, the attraction basin is filled by several ABSs. This phenomenon has the same affect as the huge number of peaks in high dimensional space.

It should be noted that this is not a problem in radius-based method, because we can manually choose a suitable radius value. As in DSI (see chapter 1), without the consideration of q (which has the same function as N_{memory}), a big value of radius creates a small number of seeds, on the contrary, a small value of radius creates a big number of seeds. However, ABSE uses the actual information of fitness landscapes, so the estimated ABS may be very small. ABSE does not have a way to control estimated ABS like tuning radius in radius-base method.

4.3 Seed Importance Calculation

In this paper, I calculate a new variable *importance* to determine if a seed is more likely to be a global peak. The method is named as Seed Importance

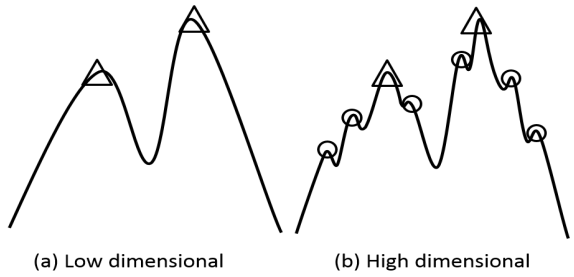


Figure 4.1: Difference between high and low dimensional space

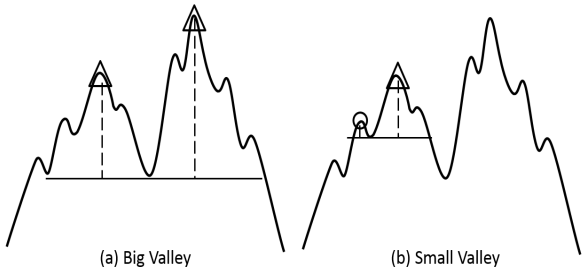


Figure 4.2: Big and small valleys

Calculation (SIC). The method is based on the assumption that the valley (detected by DMM) between two important peaks (as shown in Fig.4.2.a) is deeper than that between two unimportant peaks or an important peak and an unimportant peak (as shown in Fig.4.2.b).

I calculate an *importance* for each seed. Higher importance value means a seed is more important. To calculate the importance of a seed A, we need to test A with several other seeds.

In each test, I calculate a variable *ValleyHeight*. Assuming A is tested with B, to calculate ValleyHeight H(A,B), first I check if A and B are multi-modal, and if A is worse than B. If both conditions are satisfied, I calculate

H(A,B) by Eq.4.1. Otherwise, the ValleyHeight is set to infinite. The importance is the smallest ValleyHeight among all tests.

$$H(A,B) = max(A.fitness, B.fitness) - valley.fitness \tag{4.1}$$

Fig.4.3 gives an example of how to calculate importance. The peaks 1, 2 and 3 are multimodal with each other. I assume the lowest points on the lines between 1 and 2, 2 and 3, 1 and 3 are found by DMM as valleys.

The fitness of the peak *1* is lower than both peaks *2* and *3*, so two ValleyHeight H(1,2), H(1,3) are calculated. The importance of the peak *1* is H(1,2), because it is smaller than H(1,3).

The peak *2* is better than the peak *1*, but worse than the peak *3*, so one ValleyHeight H(2,3) is calculated, which becomes the importance of the peak *2*.

The peak *3* is the best one, so I do not calculate ValleyHeight for it. Its importance is infinite. The height of H(1,2), H(1,3), H(2,3) are shown in the Fig. 4.3.

According to the importance, the peak *3* is the most important one; the peak *2* is the second important one; the peak *1* is least important one.

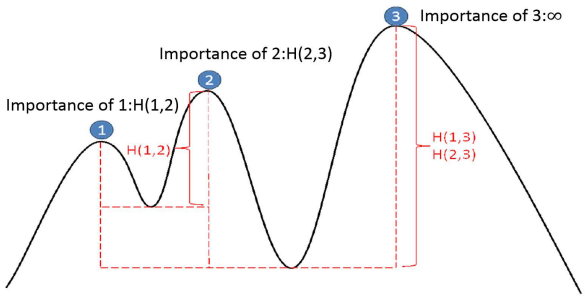


Figure 4.3: How to calculate importance

4.4 ABSEGA2 Algorithm

I integrate the SIC into ABSEGA and make a new algorithm ABSEGA2. To do this, I add an matrix I to Seed Archive to record ValleyHeight of each test. So the SIC can use I to calculate importance. The seed deletion is modified to delete seeds by importance instead of by fitness.

The proposed algorithm ABSEGA2 is presented in algorithm 4.1. The modified parts are shown in *italic*. As ABSEGA, Seed Archive SA is initialized to empty. At generation t, ABSEGA2 first selects seeds from population $P(t-1)$. Second, I make mutual tests between every new seeds and every old seeds. All points evaluated in this step are stored in a set S. This step is different from ABSEGA, in which internal tests are also used. Next, ABSs are updated using ASBE, and importance are calculated using SIC. Next, I delete covered seeds and seeds ranked, by their importance, below N_{memory}. This is the main difference between ABSEGA2 and ABSEGA in which fitness are used to delete seeds. Next, I run seed conservation and create P'

by combine the population $P(t-1)$ and the set S. The P' is divided into niches by assigning each member of P' to its nearest seed. Finally, ABSEGA2 generates the population $P(t)$ as ABSEGA.

The evaluations of interior points at a single generation is $n_{new} \cdot n_{old} \cdot p_{interior}$, where $p_{interior}$ is the number of interior points. n_{new} and n_{old} are the number of newly identified seeds and the number of old seeds in Seed Archive respectively.

Algorithm 4.1 ABSEGA2

$t = 0$
SA=$\emptyset$; - Seed Archive
Initialize $P(t)$;
Evaluate $P(t)$;
while not terminate condition **do**
  Seed Selection to SA;
  Mutual Test new seeds and old seeds in SA and store evaluated points in S.
  ABS Refining on SA;
  Seed Importance Calculation on SA;
  Seed Deletion on SA by importance;
  Seed Conservation to $P(t-1)$;
  $P' = P(t-1) \cup S$;
  Reproduce $P(t)$ through P';
  Evaluate $P(t)$;
end while

4.5 Experiments

4.5.1 Benchmark Tests

I compare our method with 4 state-of-the-art algorithms on benchmark tests used in the previous chapter. Those algorithms includes:

- Nearest Neighbor Differential Evolution Algorithm (DE)[9].
- Dynamic Archive Niching Differential Evolution Algorithm (dADE)[8].
- Niching the CMA-ES via Nearest-Better Clustering (NEA2)[22].
- CMA-ES[20] with a simple archive.

For ABSEGA2, I use the same parameter settings as ABSEGA. For DE, I use a population of 100, the recombination factor (CR) and mutation factor (F) are set to 0.5 and 0.9 respectively. For dADE, I set the parameters according to the original paper[8], except the population size and ε are set to 100 and 0.01 respectively. For NEA2, I use a population of 100 and set other parameters according to paper[22]. For CMA-ES with a simple archive, I set λ and μ to 10 and 5 respectively. I archive and restart a CMA-set when σ is smaller than 0.01. The maximum generations for each test function is the same as the previous chapter. However, the maximum generations of CMA-ES with a simple archive is 10 times bigger than other algorithms.

Table 4.1: PR of benchmark tests

Functions	ABSEGA	ABSEGA2	dADE	DE	NEA2	CMA-ES
F1	0.96	1.00	0.90	0.70	1.00	1.00
F2	0.92	1.00	1.00	0.67	1.00	1.00
F3	1.00	0.98	1.00	1.00	0.76	0.91
F4	0.95	1.00	1.00	1.00	1.00	1.00
F5	0.91	1.00	1.00	0.00	0.91	0.00
F6(2D)	0.72	1.00	1.00	1.00	0.73	0.82
F6(10D)	0.00	0.00	0.00	0.00	0.00	0.00
F7	0.80	1.00	0.00	0.00	0.00	0.00

Table 4.2: PA of benchmark tests

Functions	ABSEGA	ABSEGA2	dADE	DE	NEA2	CMA-ES
F1	0.772	0.07	1.49	0.63	0.06	0.02
F2	0.416	0.01	0.00	0.09	0.01	0.00
F3	0.053	3.98	3.03	2.09	31.78	8.13
F4	0.163	0.01	0.05	0.10	0.04	0.01
F5	0.158	0.08	0.01	37.15	9.97	8.06
F6(2D)	206	3.74	14.55	0.00	61.31	176.89
F6(10D)	451	677.67	200.21	500.00	367.42	261.11
F7	0.106	0.19	0.27	0.58	0.78	0.65

Their peak ratio (PR), peak accuracy (PA) and distance accuracy (DA) are shown in Table.4.1, Table.4.2 and Table.4.3 respectively, where CMA-ES represents CMA-ES with a simple archive. According to PR, ABSEGA2 is better than ABSEGA on function F1, F2, F4, F5, F6(2D) and F7, and slight worse on F3. Considering other state-of-the-art algorithms, dADE and CMA-ES with a simple archive obtain a zero PR on function F5, and all of them obtain a zero PR on F7. ALL algorithms obtain a zero on F6(10D). To conclude, ABSEGA2 has a better performance than other algorithms on most test functions.

Table 4.3: DA of benchmark tests

Functions	ABSEGA	ABSEGA2	dADE	DE	NEA2	CMA-ES
F1	1.363	0.27	0.17	0.71	0.09	0.08
F2	0.408	0.22	0.10	1.02	0.14	0.08
F3	0.016	0.59	0.54	0.43	5.06	1.30
F4	0.187	0.09	0.22	0.36	0.23	0.08
F5	0.026	0.02	0.01	2.31	0.36	2.80
F6(2D)	0.392	0.56	1.22	0.59	0.05	0.58
F6(10D)	12.56	21.02	11.67	21.24	20.19	19.43
F7	4.137	1.03	6.91	9.05	14.29	14.37

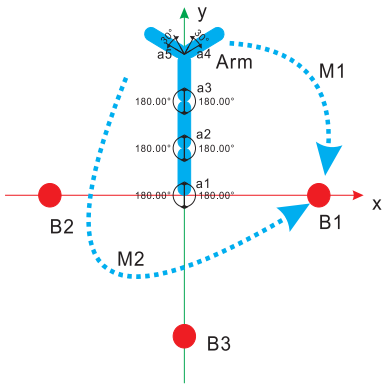


Figure 4.4: The ball catching task

4.5.2 Robot Arm

Experiments Setup

In the robot arm experiment, an ANN should be optimized to control a robotic arm to catch balls, as shown in Fig.4.4. The robotic arm is composed by connecting five rectangles. Three of them consist the stem of the arm. The others are the two fingers of the arm. The three stem parts can rotate in 360 degree. The fingers can rotate from 30 to 60 or -60 to -30 degree. Each rectangle is 1m long. The arm is placed at the origin and three balls

are placed 3m from the origin at different directions. An ANN controls the arm to catch the balls. The inputs of the ANN are angle a_1-a_5 and the vectors from the two fingers to the target ball. a_1-a_5 indicate the relative angles between the rectangles. The outputs of the ANN o_1-o_5 are angles that indicate the rotation of each rectangle in the next simulation time step.

The angles a_1-a_5 are updated by:

$$a_i = a_i + o_i, \begin{cases} a_i \in [-180, 180] & i = 1, 2, 3 \\ a_i \in [30, 60] & i = 4 \\ a_i \in [-60, -30] & i = 5 \end{cases} \tag{4.2}$$

If a_4 and a_5 exceed one of their boundaries, they will be set to the corresponding boundary. However, for a_1-a_3, they will be converted to the corresponding value that satisfies the range. For example, 210 is converted to -150. o_i is in the range $[-5, 5]$, where the positive value refers to the clockwise rotation.

The objective is to catch all three balls. To catch a ball is called as a sub-task. A sub-task has 50 time steps. When the time step of a sub-task is over, the arm is set to the initial state. Hence, the total time steps is 150. As shown in Fig.4.4, I set a_1-a_3 to 0, and a_4-a_5 to 30 and -30 respectively as the initial state of the arm.

The fitness of this task is calculated by Eq.4.3. $H_1(t)$, $H_2(t)$ are positions

of two fingers at the t step. B_i is the position of the i-th target ball. D is the distance between its two arguments. Because there are 3 balls, n is 3.

$$f = \min_{i=0}^{n}\{-\sum_{t=0}^{50}[D(H_1(t), B_i) + D(H_2(t), B_i)]\} \tag{4.3}$$

Because the three stem parts can rotate in 360 degree, this task is deceptive, which means they have a lot of local peaks. For example, to catch ball *B1*, the arm can follow either motion *M1* or *M2*. Hence, using the current fitness function, an algorithm with the multimodal optimization ability can increase the possibility to find a good solution.

I introduce a method to analyze the diversity of the population. The values of o_1, o_2, o_3 over a single sub-task at each simulation step are summed independently. The positive summed value is mapped to boolean value 1, and the negative value is mapped to 0. The three boolean values are combined to a 3 bits vector, named as a *feature vector*. It indicates the motion of the arm in a sub-task. There are 8 possible kinds of feature vector, from 000 to 111. I collect the feature vectors of all sub-tasks that individuals performed. For each feature vector kind, I count its appearance in the whole population. If it appears more than a threshold $0.15 * \textit{population size}$ times, this feature vector kind is considered *found*. I calculate the number of *found* feature

vectors as a measurement of diversity. This process is shown in Algorithm 4.2, where N is the population size.

Algorithm 4.2 Diversity Calculation

```
Input: a population;
Output: diversity;

Set D[000],....,D[111] to 0.
for each sub-task do
  for every p in the population do
    Set the arm to the initial state;
    Set the target ball;
    SO_i=0,i=1,2,3;
    for t=0 to 50 do
      Calculate o_i;
      SO_i=SO_i+o_i,i=1,2,3;
      Update the arm;
    end for
    feature vector fb = [bool(SO_1), bool(SO_2), bool(SO_3)];
    D[fb] = D[fb] + 1;
  end for
end for
diversity=0;
for each feature vector kind fb do
  if D[fb] > 0.15*N then
    diversity=diversity+1;
  end if
end for
```

I show an example of calculating diversity. Table 4.4 shows the calculation of *feature vectors* for one individual. o_1-o_3 of 3 sub-tasks are shown in the top. The summed values of o_1-o_3 are shown in row "total". The summed values are converted to boolean values as shown in row "feature vector". Because the task includes 3 sub-tasks, one individual has 3 *feature vectors*.

Table 4.4: Feature vectors of an individual

	sub-task1			sub-task2			sub-task3		
simulation time	o_1	o_2	o_3	o_1	o_2	o_3	o_1	o_2	o_3
0	3.1	-4.6	1.1	-5.6	-1.1	1.4	1.5	1.6	3.3
1	4.3	3.5	-3.7	-2.6	-5.1	4.4	3.7	2.9	1.1
...	...	...	...	...	...	...	...	...	...
49	-1.6	3.8	2.3	4.7	3.5	-2.1	4.8	-5.9	-1.3
50	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
total	-90.1	10.6	-50.7	100.3	150.6	-10.8	90.7	-100.6	100.3
feature vector	0	1	0	1	1	0	1	0	1

Table 4.5: Feature vectors of all individuals in the population

individual	sub-task1			sub-task2			sub-task3		
A	0	1	0	1	1	0	1	0	1
B	0	1	0	0	1	0	0	1	1
C	0	0	1	1	1	0	1	0	1
D	0	1	1	1	0	0	1	0	1
E	0	1	0	1	1	0	1	0	0

The *feature vectors* of all individuals in the population are shown in Table 4.5. The contents of the set D is shown in Table 4.6. The threshold is 0.75(0.15*5), so the diversity of this population is 6.

Table 4.6: The contents of the set D

D[000]	0
D[001]	1
D[010]	4
D[011]	2
D[100]	2
D[101]	3
D[110]	3
D[111]	0

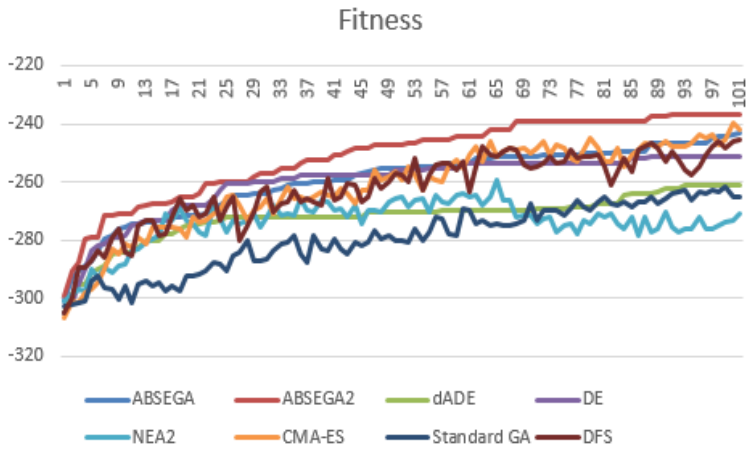


Figure 4.5: Fitness of 8 algorithms

Performance Comparison

The population size, N_{memory}, N_{expand}, crossover and mutate rate of ABSEGA2 is 100, 10, 5, 0.6 and 0.1 respectively. The number of interior points used is 1. The results are compared with ABSEGA, Standard GA[10], DFS[5] and those state-of-the-art algorithms. For Standard GA and DFS, they use the same population size, crossover and mutation rate as ABSEGA2. Each algorithm is run for 100 generations (CMA-ES with a simple archive is 1000). The results are the average of 30 runs.

The average evaluations for interior points during a single generation is 29.516, where the population size is 100. This means the algorithm spends about 30% additional time for evaluating interior points.

The fitness and diversity of 8 algorithms are shown in Fig. 4.5, Fig. 4.6 respectively, where x-axis is the number of generations and y-axis is

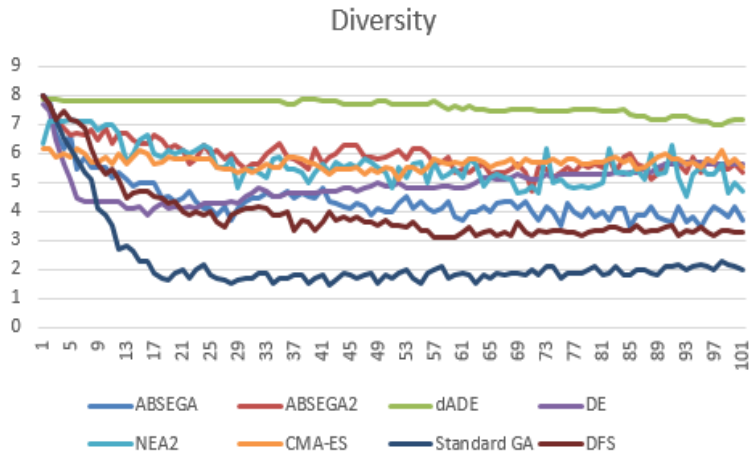


Figure 4.6: Diversity of 8 algorithms

the fitness or diversity. For ABSEGA2, it obtains the highest fitness on all tasks. The diversity of ABSEGA2, about 5.5, is ranked in the middle of all algorithms. The Standard GA obtains relatively low fitness values. Its diversity is the worst among all algorithms. DFS performs better than the Standard GA, but its diversity is the second worst among all algorithms. The dADE can keep the most diversity on all tasks, but its fitness is relative low among all algorithms. DE and CMA-ES with a simple archive have similar fitness and diversity values. Both of them are ranked in the middle among all algorithms. NEA2 can keep a relatively better diversity, but its fitness is the worst among all algorithms.

The above results count all individuals in the population. I only count the individuals that are better than three levels: -280, -320, -380, and show their diversity in Table. 4.7. The dADE can keeps the most diversity on all

Table 4.7: Diversity at different levels

Levels	-280	-320	-380
ABSEGA	1.91	2.82	3.45
ABSEGA2	3.363	4.45	5.63
dADE	3.91	6.18	6.91
DE	3.36	6.00	5.55
NEA2	1.81	3.36	4.45
CMA-ES	2.91	5.27	5.55
Standard GA	1.64	1.73	2.00
DFS	2.27	2.91	3.27

three levels. However, its best fitness is relatively low according to Fig. 4.5. DE and ABSEGA2 has similar diversity on all three levels. CMA-ES with a simple archive is worse than DE and ABSEGA2, but better than the rest of algorithms. ABSEGA, NEA2, Standard GA, DFS can only keep a low diversity on all levels.

To conclude, ABSEGA2 can find a better solution and keep a good diversity in the population. ABSEGA is worse than ABSEGA2 on both fitness and diversity. Because the difference between ABSEGA2 and ABSEGA is the use of importance, so the importance plays an important role on keeping diversity and finding better solutions.

4.6 Conclusion

The ABSEGA2 is an improvement of ABSEGA for Neuroevolution tasks. A newly introduced variable importance is used as a measurement to differ

which seed is more important, so the algorithm can track important peaks among even hundreds of unimportant peaks on Neuroevolution tasks.

I first compared ABSEGA2 with some state-of-the-art algorithms including dADE, DE, CMA-ES with a simple archive, and NEA2 on benchmark tests. Second, I examined ABSEGA2 in a robotic arm problem, in which I evolved an artificial neural network (ANN) to control the arm to catch balls. I compared ABSEGA2 with a radius-based method Dynamic Fitness Sharing (DFS), the Standard Genetic Algorithm and those state-of-the-art algorithms. The results showed that ABSEGA2 has a high ability to escape local optimal and find relatively better solutions. The experiments also showed that the calculation of Importance plays an important role on keeping diversity on complex tasks.

Chapter 5

Attraction Basin Sphere Estimation CMA-ES

5.1 Introduction

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES)[20] is an optimization method which is capable of dealing with many difficulties that can occur in an optimization task. Due to the nature of CMA-ES, it is hard to design a niching method for a single CMA-ES population. Hence, niching CMA-ES methods are usually a hybridization of running multiple CMA-ES instances in parallel with a high-level control strategy[27]. The control strategy dynamically adjusts the instances to avoid multiple instances searching in the same area[22, 28].

CMA-ES is an ideal mechanism to work with ABSE. Because, in CMA-ES, offspring are generated by muting a search point, so the search point can be considered as a representative of its offspring. Hence, I can create niches only according to search points. This feature decreases the computation on evaluating interior points, therefore largely improves the efficiency of using ABSE. In this chapter, I combine CMA-ES and ABSE, with the hope that the combination could have a nice ability to locate multiple peaks. The SIC in ABSEGA2 are also improved and named as SIC2.

The rest of this chapter is composed as follows: Section 5.2 explains the improved Seed Importance Calculation2 (SIC2). Section 5.3 describes the Attraction Basin Sphere Estimation CMA-ES Algorithm (ABSE-CMA-ES). Section 5.4 describes the experiments for testing our algorithm. Section 5.5 concludes this study with some remarks.

5.2 Seed Importance Calculation2

The improved SIC2 is based on the assumption proposed in Nearest-Better Clustering[23] that the distance from local peaks to global peaks is shorter than the distance between global peaks.

Given a point p and an array of points PS that have been tested with p. For each point i in PS, if i and p are multimodal, and i is better than p,

then I record i in a set V. After that, I find the nearest point to p in V and record the distance between them as *importance* of p. The process is shown in algorithm 5.1.

Algorithm 5.1 Seed Importance Calculation2 (SIC2)

input:
p - the point to be calculated;
PS - array of points that have been tested with p;
output:
p.importance

p.importance=∞;
$V = \emptyset$;
for all i=1..size of PS **do**
  if PS[i] and p are multimodal and
    PS[i].fitness>p.fitness **then**
    $V = V \bigcup T[i]$;
  end if
end for
if V is not $\emptyset$ **then**
  p.importance=distance between p and its nearest element in V;
end if

Fig. 5.1 shows an example. There are 2 important peaks, marked by triangle, and 4 unimportant peaks, marked by circle, located on the surface of the two important peaks. The arrow in Fig.5.1 shows the importance of each peak. The important peak on the right side does not have an arrow, so its importance is infinite. The important peak on the left side have a longer arrow than all unimportant peaks, so its importance is bigger than those unimportant peaks.

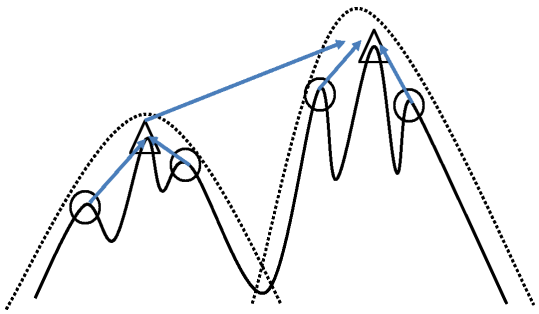


Figure 5.1: An example of seed importance calculation2

5.3 Attraction Basin Sphere Estimation CMA-ES

In Dynamic Niching with CMA [27], seeds are identified from all $\lambda \cdot (q+1)$ offspring which are generated by $q+1$ search points. Seeds are detected in a different way in our method. Because CMA-ES is a local search algorithm, in most of the time, offspring of a search point are generated around it. So a search point can be considered as a representative of its offspring. If we keep search points tracking different peaks, their offspring also track different peaks. In another word, we only need to check search points, and keep them tracking different peaks. Because the number of search points is much smaller than the number of offspring, this idea makes the algorithm more efficient. The proposed algorithm, Attraction Basin Sphere Estimation CMA-ES (ABSE-CMA-ES), is shown in algorithm 5.2.

Given q, q search points associated with q CMA-sets are initialized. Each

search point is also associated with a variable $\lambda_{niching}$ which is initialized to a parameter λ. Seed Archive SA is initialized to empty.

Until stopping criteria are met, the following procedure takes place. First, each search point generates in every generation $\lambda_{niching}$ samples (offspring) based on its CMA-set. Second, the fitness of offspring are evaluated. Third, I select all uncovered search points as seeds. It should be noted that this step is different from ABSEGA and ABSEGA2, in which I only select up to N_{expand} seeds. Forth, I check if each seed has been tested with its nearest seed in SA. If not, a test is made and the result is recorded in Seed Archive. Because seeds are inserted and deleted continuously at each generation, the nearest seed may change during the optimization process. So following the optimization process, a seed will be tested with more and more points, and its ABS will become more and more accuracy. It should be noted that this step is also different from ABSEGA2. Fifth, I update ABS and importance of all seeds using ABSE and SIC2. Sixth, I delete covered seeds and seeds ranked, by their importance, below N_{memory}. Next, all search points are classified into niches based on their distance to seeds in Seed Archive. The $\lambda_{niching}$ of a search point is adjusted according to its rank in its niche. The adjustment is based on the principle that $\lambda_{niching}$ of higher ranked search point should be increased and $\lambda_{niching}$ of lower ranked search point should be decreased. A constraint of the adjustment is that the sum of all $\lambda_{niching}$ should be always

equal to $q \cdot \lambda$. Finally, for a search point, if its $\lambda_{niching}$ is equal to 0 or the σ of its CMA-set is small than a threshold T_σ, this search point and its CMA-set and $\lambda_{niching}$ are re-initialized. Otherwise, its CMA-set is updated and its best offspring is set as a search point of the next generation. The usage of T_σ is to provide an opportunity to explore the search space, because a single run of CMA-ES tends to concentrate rapidly on a very small search space portion.

Algorithm 5.2 ABSE-CMA-ES

```
SA=∅; - Seed Archive
initialize q search points and their CMA-sets;
λ_niching of each search point is initialized to λ;
while not terminate condition do
  for all i=1..q search points do
    Generate λ_niching[i] samples based on the CMA distribution of i;
  end for
  Evaluate Fitness of the population;
  Select all uncovered search points to SA;
  Test each seed with its nearest seed in SA;.
  Update ABS and importance on SA;
  Seed Deletion on SA by importance;
  Classify search points into niches according to SA;
  for every niches do
    Adjust λ_niching for its search points;
  end for
  for all i=1..q search points do
    if λ_niching[i]==0 or σ[i] < T_σ then
      re-initialize search point i and its CMA-set, λ_niching;
    else
      Update CMA-set i;
      Set the best offspring of search point i as a search point of the next
      generation;
    end if
  end for
end while
```

5.4 Experiments

The benchmark tests used in the previous 2 sections are not complex enough. CEC 2013 Niching Methods Competition[18] provides a benchmark set which is a common platform that encourages fair and easy comparisons across different niching algorithms. The set includes 20 test functions.

1. F1: Five-Uneven-Peak Trap (1D).
2. F2: Equal Maxima (1D).
3. F3: Uneven Decreasing Maxima (1D)
4. F4: Himmelblau (2D).
5. F5: Six-Hump Camel Back (2D)
6. F6: Shubert (2D, 3D)
7. F7: Vincent (2D, 3D)
8. F8: Modified Rastrigin - All Global Optima (2D)
9. F9: Composition Function 1 (2D)
10. F10: Composition Function 2 (2D)
11. F11: Composition Function 3 (2D, 3D, 5D, 10D)

12. F12: Composition Function 4 (3D, 5D, 10D, 20D)

We show function F1-F8 as follows. Please see CEC 2013 [18] for F9-F12.

$$F1(x) = \begin{cases} 80(2.5-x) & for\ 0 \le x \le 2.5, \\ 64(x-2.5) & for\ 2.5 \le x \le 5.0, \\ 64(7.5-x) & for\ 2.5 \le x \le 5.0, \\ 28(x-7.5) & for\ 2.5 \le x \le 5.0, \\ 28(17.5-x) & for\ 2.5 \le x \le 5.0, \\ 32(x-17.5) & for\ 2.5 \le x \le 5.0, \\ 32(27.5-x) & for\ 2.5 \le x \le 5.0, \\ 80(x-27.5) & for\ 2.5 \le x \le 5.0. \end{cases} x \in [0, 30] \tag{5.1}$$

$$F2(x) = \sin^6(5\pi x), x \in [0, 1] \tag{5.2}$$

$$F3(x) = \exp(-2\log(2)(\frac{x-0.08}{0.854})^2)\sin^6(5\pi(x^{3/4}-0.05)), x \in [0, 1] \tag{5.3}$$

$$F4(x, y) = 200 - (x^2 + y - 11)^2 - (x + y^2 - 7)^2, x, y \in [-6, 6] \tag{5.4}$$

$$F5(x, y) = -4[(4-2.1x^2+\frac{x^4}{3})x^2+xy+(4y^2-4)y^2], x \in [-1.9, 1.9], y \in [-1.1, 1.1] \tag{5.5}$$

$$F6(\vec{x}) = -\prod_{i=1}^{D}\sum_{j=1}^{5} j\cos[(j+1)x_i + j], x_i \in [-10, 10] \tag{5.6}$$

$$F7(\vec{x}) = \frac{1}{D}\sum_{i=1}^{D}\sin(10\log(x_i)), x_i \in [0.25, 10] \tag{5.7}$$

Table 5.1: Test Functions

Function	Peaks	Height	r
F1(1D)	2	200.0	0.01
F2(1D)	5	1.0	0.01
F3(1D)	1	1.0	0.01
F4(2D)	4	200.0	0.5
F5(2D)	2	1.03163	0.5
F6(2D)	18	186.731	0.5
F7(2D)	36	1.0	0.2
F6(3D)	81	2709.0935	0.5
F7(3D)	216	1.0	0.2
F8(2D)	12	-2.0	0.01
F9(2D)	6	0	0.01
F10(2D)	8	0	0.01
F11(2D)	6	0	0.01
F11(3D)	6	0	0.01
F12(3D)	8	0	0.01
F11(5D)	6	0	0.01
F12(5D)	8	0	0.01
F11(10D)	6	0	0.01
F12(10D)	8	0	0.01
F12(20D)	8	0	0.01

$$F8(\vec{x}) = -\sum_{i=1}^{D}(10 + 9\cos(2\pi k_i x_i)), x_i \in [0,1], k_1 = 3, k_2 = 4 \qquad (5.8)$$

The index, function, the number of global peaks and global peak height are shown in Table. 5.1.

At the end of an optimization run, I calculate the number of found global peaks (NPF) in this way: First, I need to specify a level of *accuracy* (typically $0 < \epsilon < 1$), and a radius r. Next, I check all individuals (for our method, Seed Archive are also included) sequentially, if an individual has a fitness that is within ϵ to the global peak height, and it is not within r of any recorded

Table 5.2: MaxFEs of test functions

Functions	MaxFEs
F1 to F5 (1D or 2D)	5.0E+04
F6 to F11 (2D)	2.0E+05
F6 to F12 (3D or higher)	4.0E+05

solutions, then it is recorded as a *solution*. Finally, the number of recorded solutions is *NPF*. The radius r depends on test functions, as shown in Table. 5.1.

I use peak ratio (PR) and success rate (SR)[18] as two performance measures to evaluate the performance of a niching algorithm over multiple runs. PR measures the average percentage of global peaks found over multiple runs (see chapter 3). SR measures the percentage of successful runs (a successful run is defined as a run where all global peaks are found) out of all runs:

$$SR = \frac{NSR}{NR} \tag{5.9}$$

where NSR denotes the number of successful runs and NR is the number of runs. The maximum number of function evaluations (MaxFEs) for different test functions are shown in Table. 5.2.

All algorithms are tested on the functions described above. The number of runs (NR) is set to 50. I use a *level of accuracy* ϵ of 1.0E-02. The algorithm terminates when reaching MaxFEs. In ABSE, I generate only *one* interior point which is in the middle of the line between two given points.

When a new optimization algorithm proposal is developed, it is necessary to compare it with previous approaches. Following Puris et al.[24] and Derrac et al.[7], I use some nonparametric statistical tests in this paper to find significant differences among the results obtained by the studied methods. The statistical analyses used in this paper include Friedman's test, Iman and Davenport's test, Holm's test, Wilcoxon's signed-ranks test and Contrast Estimation.

Friedman's test can be used to rank algorithms according to their performance. Iman and Davenport's test may be used to see whether there are significant statistical differences among the algorithms in certain groups (three or more algorithms). If differences are detected, then Holm's test is employed to compare the best ranking algorithm (control algorithm) with the remaining ones. With Wilcoxon's signed-ranks test, the results of two algorithms may be directly compared. Contrast Estimation can be used to estimate the difference between two or more algorithms.

5.4.1 Internal Analysis of ABSE-CMA-ES

Size of Peaks to Identify and Search Points

Before I compare the proposed algorithm with other niching methods, I would like to analyse the effect of the parameters. Except the parameters of CME-

ES, the algorithm has two parameters: the size of Seed Archive N_{memory} and search points q. I run the algorithm with different combinations of N_{memory} and q and check the differences on the results. For both N_{memory} and q, three values are chosen according to the NKP of a test function respectively. They are $N_{memory} = NKP, 2.5NKP, 5NKP$ and $q = 0.5NKP, 2.5NKP, 5NKP$ (N_{memory}(x) refers to N_{memory}=x·NKP, q(x) has the same meaning). Table. 5.3 shows parameters of CMA-ES: initial offspring number of a search point (λ), initial step size (σ) and re-initializing threshold of σ (T_σ) which is set according to the *level of accuracy* ϵ. Other parameters of CMA-ES are set as described in [20]. The PR and SR of the results are shown in Table.5.4 and Table.5.5. It should be noted that in a real world task NKP is hard to be known, however usually we have a desired number of solutions which can be considered as NKP.

I consider the effect of N_{memory} and q separately. The nonparametric tests are applied to analyse the PR of the results. I present in Table.5.6 the Friedman's test calculated on F6(2D)-F12(20D) for different values of N_{memory} and q. Because all parameter settings obtain the same results on F(1D)-F5(2D), they are not included in the Friedman's test. In fact, all experiments in this paper obtain the same results on F(1D)-F5(2D), so they are not included in any analysis. The results of Friedman's test show that $N_{memory}(5)$ and $q(5)$ are ranked as the best parameter setting respectively

in their group. However, the results of ImanDavenport's test show that the N_{memory} and q could not produce significant differences at a level of significance α=0.05, which means parameter settings are not so important to obtain a high performance.

To continue, Holm's test is applied to compare the best rank parameter setting with each of the two remaining ones. For this test, the settings are ordered in descending order according to rank. Table.5.7 contains all the computations associated with Holm's procedure. On both N_{memory} and q groups, the control parameter settings do not perform significant better than the corresponding parameter settings (second column). This conclusion is arrived at because the p-values are bigger than the corresponding α/i values. However, the p-values corresponding to N_{memory} is smaller than the p-values corresponding to q, which means N_{memory} has more influence on the performance.

Moreover, I apply Wilcoxon's Signed-Rank test to compare the best rank parameter setting with each of the two remaining ones for each test function. Table.5.8 shows the values of R^+ and R^-, together with the p-values computed for this test. The mark '+' or '-' placed before N_{memory} or q means whether the control parameter setting is significant better or worse than the corresponding parameter setting at a level of significance α=0.05. On N_{memory}, as expected, the number of '+' is much more than '-' (12 and

Table 5.3: Parameters for ABSE-CMA-ES

Parameter	λ	σ	T_σ
Value	10	1	1.0E-04

Table 5.4: Results of PR of different parameter settings

Functions	$N_m.(1)$ q(0.5)	$N_m.(1)$ q(2.5)	$N_m.(1)$ q(5)	$N_m.(2.5)$ q(0.5)	$N_m.(2.5)$ q(2.5)	$N_m.(2.5)$ q(5)	$N_m.(5)$ q(0.5)	$N_m.(5)$ q(2.5)	$N_m.(5)$ q(5)
F1(1D)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
F2(1D)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
F3(1D)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
F4(2D)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
F5(2D)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
F6(2D)	0.89	0.95	0.64	0.61	0.68	0.61	0.99	1.00	0.97
F7(2D)	0.52	0.50	0.59	0.77	0.71	0.74	0.98	0.97	0.96
F6(3D)	0.25	0.24	0.19	0.21	0.23	0.17	0.20	0.20	0.18
F7(3D)	0.44	0.34	0.34	0.58	0.55	0.47	0.73	0.70	0.69
F8(2D)	0.83	1.00	1.00	0.83	1.00	1.00	0.84	1.00	1.00
F9(2D)	0.73	0.92	1.00	0.77	1.00	1.00	0.79	1.00	1.00
F10(2D)	0.34	0.72	0.81	0.36	0.80	0.86	0.38	0.74	0.85
F11(2D)	0.47	0.83	0.92	0.48	0.96	0.95	0.52	0.94	0.95
F11(3D)	0.65	0.73	0.71	0.58	0.67	0.71	0.62	0.70	0.71
F12(3D)	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.69	0.75
F11(5D)	0.65	0.65	0.56	0.64	0.65	0.58	0.64	0.62	0.62
F12(5D)	0.68	0.70	0.74	0.72	0.75	0.74	0.67	0.73	0.74
F11(10D)	0.36	0.33	0.35	0.35	0.29	0.32	0.41	0.30	0.30
F12(10D)	0.56	0.49	0.50	0.52	0.50	0.51	0.51	0.60	0.56
F12(20D)	0.42	0.33	0.38	0.43	0.40	0.51	0.48	0.50	0.50

1 respectively). However, on q, the number of '+' is nearly equal to '-' (10 and 9 respectively). The '-' on F6(2D)-F7(3D) can be explained by NKP are very large on those functions, so a big value of q leads to, due to the MaxFEs are fixed, the algorithm does not have enough generations to search the landscapes.

Those results suggest that q and N_{memory} have relation to the NKP of the problem. Big values of q and N_{memory} can lead to a better performance, but not in a significant way.

ABSE and SIC2

The algorithm has two important components:ABSE and SIC2. In this experiment, I analyse their contribution to the whole algorithm. First, I want

Table 5.5: Results of SR of different parameter settings

Functions	$N_{m.}(1)$ q(0.5)	$N_{m.}(1)$ q(2.5)	$N_{m.}(1)$ q(5)	$N_{m.}(2.5)$ q(0.5)	$N_{m.}(2.5)$ q(2.5)	$N_{m.}(2.5)$ q(5)	$N_{m.}(5)$ q(0.5)	$N_{m.}(5)$ q(2.5)	$N_{m.}(5)$ q(5)
F1(1D)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
F2(1D)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
F3(1D)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
F4(2D)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
F5(2D)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
F6(2D)	0.18	0.27	0.00	0.00	0.00	0.00	0.82	1.00	0.91
F7(2D)	0.00	0.00	0.00	0.09	0.00	0.00	0.36	0.09	0.27
F6(3D)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
F7(3D)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
F8(2D)	0.00	1.00	1.00	0.00	1.00	1.00	0.00	1.00	1.00
F9(2D)	0.00	0.55	1.00	0.00	1.00	1.00	0.00	1.00	1.00
F10(2D)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
F11(2D)	0.00	0.27	0.55	0.00	0.36	0.73	0.00	0.64	0.73
F11(3D)	0.00	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00
F12(3D)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
F11(5D)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
F12(5D)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
F11(10D)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
F12(10D)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
F12(20D)	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 5.6: Results of Friedman's Test of different parameter settings

Parameter	Ranking	Parameter	Ranking
$N_{memory}(1)$	2.267E+00	q(0.5)	2.200E+00
$N_{memory}(2.5)$	2.200E+00	q(2.5)	2.000E+00
$N_{memory}(5)$	1.533E+00	q(5)	1.800E+00
Iman and Davenport	2.755E+00	Iman and Davenport	5.833E-01
P-value	8.085E-02	P-value	5.647E-01

Table 5.7: Holm's Test for $\alpha = 0.05$ of different parameter settings

i	$N_{memory}(5)$ vs	$z = (R_0 - R_i)/SE$	p-value	Holm
2	$N_{memory}(1)$	2.008E+00	4.461E-02	0.025
1	$N_{memory}(2.5)$	1.826E+00	6.789E-02	0.05
i	q(5) vs	$z = (R_0 - R_i)/SE$	p-value	Holm
2	q(0.5)	1.095E+00	2.733E-01	0.025
1	q(2.5)	5.477E-01	5.839E-01	0.05

Table 5.8: Results of Wilcoxon's Signed-Rank Test of different parameter settings

	$N_{memory}(5)$ vs	R^+	R^-	p-value	q(5) vs	R^+	R^-	p-value
F6(2D)	$+N_{memory}(1)$	406	0	0.0000	-q(0.5)	15	157	0.0010
	$+N_{memory}(2.5)$	561	0	0.0000	-q(2.5)	18	259	0.0001
F7(2D)	$+N_{memory}(1)$	561	0	0.0000	q(0.5)	254	182	0.2177
	$+N_{memory}(2.5)$	560	1	0.0000	+q(2.5)	247	29	0.0005
F6(3D)	$-N_{memory}(1)$	39	340	0.0002	-q(0.5)	22	414	0.0000
	$N_{memory}(2.5)$	136	270	0.0630	-q(2.5)	24	505	0.0000
F7(3D)	$+N_{memory}(1)$	561	0	0.0000	-q(0.5)	1	560	0.0000
	$+N_{memory}(2.5)$	561	0	0.0000	-q(2.5)	60	405	0.0002
F8(2D)	$N_{memory}(1)$	0	0	N/A	+q(0.5)	561	0	0.0000
	$N_{memory}(2.5)$	0	0	N/A	q(2.5)	0	0	N/A
F9(2D)	$+N_{memory}(1)$	82	9	0.0054	+q(0.5)	561	0	0.0000
	$N_{memory}(2.5)$	0	0	N/A	+q(2.5)	15	0	0.0253
F10(2D)	$N_{memory}(1)$	147	85	0.1401	+q(0.5)	561	0	0.0000
	$N_{memory}(2.5)$	72	119	0.1711	+q(2.5)	279	21	0.0001
F11(2D)	$+N_{memory}(1)$	124	48	0.0495	+q(0.5)	528	0	0.0000
	$N_{memory}(2.5)$	53	53	0.5000	q(2.5)	67	24	0.0668
F11(3D)	$N_{memory}(1)$	12	34	0.1846	+q(0.5)	190	0	0.0001
	$N_{memory}(2.5)$	36	10	0.1136	q(2.5)	52	39	0.3264
F12(3D)	$N_{memory}(1)$	0	0	N/A	q(0.5)	0	0	N/A
	$N_{memory}(2.5)$	0	0	N/A	q(2.5)	0	0	N/A
F11(5D)	$N_{memory}(1)$	30	25	0.4013	-q(0.5)	36	135	0.0154
	$N_{memory}(2.5)$	64	56	0.4091	-q(2.5)	36	135	0.0154
F12(5D)	$N_{memory}(1)$	37	29	0.3594	+q(0.5)	73	6	0.0043
	$N_{memory}(2.5)$	15	40	0.1020	q(2.5)	15	6	0.3173
F11(10D)	$N_{memory}(1)$	20	17	0.8295	-q(0.5)	7	60	0.0091
	$N_{memory}(2.5)$	76	30	0.0749	q(2.5)	21	7	0.2059
F12(10D)	$+N_{memory}(1)$	145	45	0.0222	q(0.5)	60	77	0.3300
	$+N_{memory}(2.5)$	137	34	0.0126	q(2.5)	39	52	0.3264
F12(20D)	$+N_{memory}(1)$	422	14	0.0000	q(0.5)	54	24	0.1190
	$+N_{memory}(2.5)$	92	13	0.0066	+q(2.5)	78	0	0.0011

to disable the ABSE component. So I stop checking cover to exclude the contribution of ABSE to the algorithm. So only SIC2 contribute to the niching results. Second, I want to disable the SIC2 component. I replace *importance* by fitness when deleting seeds, so I can examine the improvement of using SIC2 comparing to use fitness. The PR and SR results are shown in Table.5.9 and Fig.5.10 respectively, in which original, original-ABSE, original-SIC2 are the original algorithm, algorithm without using ABSE and algorithm without using SIC2 respectively.

I present in Table.5.11 the Friedman's test. The results of ImanDavenport's test show that different components could produce significant differences at a level of significance α=0.05.

Table 5.9: Results of PR of disabling different components of the algorithm

Functions	Original	Original-ABSE	Original-SIC2
F1(1D)	1.000E+00	1.000E+00	1.000E+00
F2(1D)	1.000E+00	1.000E+00	1.000E+00
F3(1D)	1.000E+00	1.000E+00	1.000E+00
F4(2D)	1.000E+00	1.000E+00	1.000E+00
F5(2D)	1.000E+00	1.000E+00	1.000E+00
F6(2D)	1.000E+00	9.190E-01	1.000E+00
F7(2D)	9.770E-01	8.740E-01	9.920E-01
F6(3D)	2.550E-01	2.140E-01	3.230E-01
F7(3D)	7.330E-01	3.880E-01	6.770E-01
F8(2D)	1.000E+00	6.590E-01	1.000E+00
F9(2D)	1.000E+00	8.790E-01	1.000E+00
F10(2D)	8.630E-01	7.730E-01	7.610E-01
F11(2D)	9.550E-01	9.240E-01	9.850E-01
F11(3D)	7.270E-01	6.520E-01	6.670E-01
F12(3D)	7.500E-01	1.930E-01	6.590E-01
F11(5D)	6.510E-01	2.270E-01	6.210E-01
F12(5D)	7.500E-01	1.140E-01	7.270E-01
F11(10D)	4.090E-01	1.500E-02	2.880E-01
F12(10D)	6.020E-01	8.000E-02	4.660E-01
F12(20D)	5.110E-01	2.730E-01	3.640E-01

Table.5.12 shows the results of Holm's test applied to compare the best rank component with each of the two remaining ones. The control component performs significant better than the algorithm without using ABSE, but not significant better on algorithm without using SIC2. I present in Table.5.13 the Contrast Estimation, in which the detailed differences between different components are presented.

In those experiments, I can conclude that ABSE is very important to to obtain a high performance. SIC2 can improve the performance, but not in a significant way. Their combination can obtain the best performance.

Table 5.10: Results of SR of disabling different components of the algorithm

Functions	Original	Original-ABSE	Original-SIC2
F1(1D)	1.000E+00	1.000E+00	1.000E+00
F2(1D)	1.000E+00	1.000E+00	1.000E+00
F3(1D)	1.000E+00	1.000E+00	1.000E+00
F4(2D)	1.000E+00	1.000E+00	1.000E+00
F5(2D)	1.000E+00	1.000E+00	1.000E+00
F6(2D)	1.000E+00	5.450E-01	1.000E+00
F7(2D)	7.270E-01	0.000E+00	6.450E-01
F6(3D)	0.000E+00	0.000E+00	0.000E+00
F7(3D)	0.000E+00	0.000E+00	0.000E+00
F8(2D)	1.000E+00	0.000E+00	1.000E+00
F9(2D)	1.000E+00	5.450E-01	1.000E+00
F10(2D)	0.000E+00	0.000E+00	0.000E+00
F11(2D)	9.090E-01	6.360E-01	7.420E-01
F11(3D)	0.000E+00	0.000E+00	0.000E+00
F12(3D)	0.000E+00	0.000E+00	0.000E+00
F11(5D)	0.000E+00	0.000E+00	0.000E+00
F12(5D)	0.000E+00	0.000E+00	0.000E+00
F11(10D)	0.000E+00	0.000E+00	0.000E+00
F12(10D)	0.000E+00	0.000E+00	0.000E+00
F12(20D)	0.000E+00	0.000E+00	0.000E+00

Table 5.11: Results of Friedman's Test of disabling different components of the algorithm

Component	Ranking
Original	1.300E+00
Original-ABSE	2.933E+00
Original-SIC2	1.767E+00
Iman and Davenport	3.391E+01
P-value	3.311E-08

Table 5.12: Holm's Test for $\alpha = 0.05$ of disabling different components of the algorithm

i	Original vs	$z = (R_0 - R_i)/SE$	p	Holm
2	Original-ABSE	4.473E+00	7.711E-06	0.025
1	Original-SIC2	1.278E+00	2.012E-01	0.05

Table 5.13: Contrast estimation of disabling different components of the algorithm

	Original	Original-ABSE	Original-SIC2
Original	0.000E+00	2.090E-01	5.900E-02
Original-ABSE	-2.090E-01	0.000E+00	-1.500E-01
Original-SIC2	-5.900E-02	1.500E-01	0.000E+00

5.4.2 Comparative Study With Others Algorithms

The best results of the proposed algorithm are compared with the best 5 niching methods in CEC 2013 Niching Methods Competition. Those algorithms includes:

- Nearest Neighbor Differential Evolution Algorithm (DE/nrand)[9].
- Dynamic Archive Niching Differential Evolution Algorithm (dADE/nrand)[8].
- Niching the CMA-ES via Nearest-Better Clustering (NEA2)[22].
- CMA-ES[20] with a simple archive.
- Niching Variable Mesh Optimization (NVMO)[19].

The PR and SR results are shown in Table.5.14 and Table.5.15 respectively, where the ABSE represents the proposed algorithm, and CMA-ES represents CMA-ES with a simple archive. I apply Wilcoxon's Signed-Rank test to compare the proposed algorithm with other algorithms presented in this section. According to the results of Wilcoxon's test summarized in Table.5.16, it can be observed that:

Table 5.14: Results of PR of the comparison with other algorithms

Functions	ABSE	dADE/nrand/1	DE/nrand/2	NEA2	CMA-ES	NVMO
F1(1D)	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00
F2(1D)	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00
F3(1D)	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00
F4(2D)	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00
F5(2D)	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00
F6(2D)	1.000E+00	1.000E+00	6.680E-01	9.630E-01	7.830E-01	9.960E-01
F7(2D)	9.770E-01	9.600E-01	2.760E-01	9.250E-01	5.290E-01	1.000E+00
F6(3D)	2.550E-01	9.900E-01	3.640E-01	2.400E-01	1.140E-01	3.000E-01
F7(3D)	7.330E-01	5.910E-01	6.500E-02	5.940E-01	2.770E-01	6.860E-01
F8(2D)	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00
F9(2D)	1.000E+00	6.670E-01	6.660E-01	9.660E-01	9.760E-01	6.670E-01
F10(2D)	8.630E-01	7.470E-01	6.270E-01	8.500E-01	7.870E-01	7.430E-01
F11(2D)	9.550E-01	6.670E-01	6.660E-01	9.700E-01	9.660E-01	6.670E-01
F11(3D)	7.270E-01	6.670E-01	6.660E-01	8.160E-01	7.500E-01	6.670E-01
F12(3D)	7.500E-01	6.420E-01	4.070E-01	7.220E-01	6.570E-01	7.230E-01
F11(5D)	6.510E-01	6.670E-01	6.660E-01	6.730E-01	6.660E-01	6.730E-01
F12(5D)	7.500E-01	4.800E-01	2.820E-01	6.950E-01	5.850E-01	4.830E-01
F11(10D)	4.090E-01	6.300E-01	5.130E-01	6.660E-01	3.400E-01	4.700E-01
F12(10D)	6.020E-01	1.170E-01	2.170E-01	6.660E-01	5.960E-01	1.330E-01
F12(20D)	5.110E-01	5.000E-03	2.300E-01	3.600E-01	4.470E-01	0.000E+00

Table 5.15: Results of SR of the comparison with other algorithms

Functions	ABSE	dADE/nrand/1	DE/nrand/2	NEA2	CMA-ES	NVMO
F1(1D)	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00
F2(1D)	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00
F3(1D)	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00
F4(2D)	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00
F5(2D)	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00
F6(2D)	1.000E+00	1.000E+00	0.000E+00	4.800E-01	2.000E-02	9.200E-01
F7(2D)	6.452E-01	1.800E-01	0.000E+00	8.000E-02	0.000E+00	1.000E+00
F6(3D)	0.000E+00	5.000E-01	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F7(3D)	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F8(2D)	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00	1.000E+00
F9(2D)	1.000E+00	0.000E+00	0.000E+00	8.000E-01	8.600E-01	0.000E+00
F10(2D)	0.000E+00	0.000E+00	0.000E+00	1.800E-01	6.000E-02	0.000E+00
F11(2D)	7.419E-01	0.000E+00	0.000E+00	8.200E-01	8.000E-01	0.000E+00
F11(3D)	0.000E+00	0.000E+00	0.000E+00	1.000E-01	2.000E-02	0.000E+00
F12(3D)	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F11(5D)	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F12(5D)	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F11(10D)	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F12(10D)	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00
F12(20D)	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00	0.000E+00

- the proposed algorithm is significant better than the dADE/nrand/1, DE/nrand/2, CMA-ES with a simple archive and NVMO at a level of significance α=0.05.

- the proposed algorithm is better than NEA2 (because R^+ is higher than R^-), but not in a significant way.

Table 5.16: Results of Wilcoxon's Signed-Rank Test of the comparison with other algorithms

ABSE vs	R^+	R^-	p-value
dADE/nrand/1	7.000E+01	2.100E+01	4.363E-02
DE/nrand/2	9.700E+01	8.000E+00	2.640E-03
NEA2	6.350E+01	4.150E+01	2.451E-01
CMA-ES	9.600E+01	9.000E+00	3.170E-03
NVMO	8.700E+01	1.800E+01	1.500E-02

5.4.3 ABSEGA, ABSEGA2 and ABSE-CMA-ES

I compare the performance of three proposed algorithm ABSEGA, ABSEGA2 and ABSE-CMA-ES on the CEC 2013 benchmark tests and the ball catching task used in chapter 4.

The PR and SR of the three algorithms on CEC 2013 are shown in Table 5.17 and Table 5.18. I present in Table.5.19 the Friedman's test. The results of ImanDavenport's test show that three algorithms have significant differences at a level of significance α=0.05.

Table.5.20 shows the results of Holm's test applied to each pair of the three algorithms. According to both Friedman's test and Holm's test, ABSE-CMA-ES performs significant better than ABSEGA and ABSEGA2 at a level of significance α=0.05. ABSEGA2 is slight better than ABSEGA but not significant. I present in Table.5.21 the Contrast Estimation, in which the detailed differences between different components are presented.

The fitness and diversity of the three algorithms on the ball catching task are shown in Fig. 5.2, Fig. 5.3 respectively. ABSE-CMA-ES has similar

Table 5.17: Results of PR of three algorithms

Functions	ABSEGA	ABSEGA2	ABSE-CMA-ES
F1(1D)	1.000E+00	1.000E+00	1.000E+00
F2(1D)	1.000E+00	1.000E+00	1.000E+00
F3(1D)	1.000E+00	1.000E+00	1.000E+00
F4(2D)	1.000E+00	1.000E+00	1.000E+00
F5(2D)	1.000E+00	1.000E+00	1.000E+00
F6(2D)	1.000E+00	1.000E+00	1.000E+00
F7(2D)	1.000E+00	9.722E-01	9.770E-01
F6(3D)	2.593E-01	8.148E-01	2.550E-01
F7(3D)	4.630E-01	3.704E-01	7.330E-01
F8(2D)	1.000E+00	1.000E+00	1.000E+00
F9(2D)	1.667E-01	5.000E-01	1.000E+00
F10(2D)	3.750E-01	3.750E-01	8.630E-01
F11(2D)	1.667E-01	3.333E-01	9.550E-01
F11(3D)	1.667E-01	3.333E-01	7.270E-01
F12(3D)	2.500E-01	2.500E-01	7.500E-01
F11(5D)	0.000E+00	1.667E-01	6.510E-01
F12(5D)	1.250E-01	1.250E-01	7.500E-01
F11(10D)	3.333E-01	3.333E-01	4.090E-01
F12(10D)	0.000E+00	0.000E+00	6.020E-01
F12(20D)	0.000E+00	0.000E+00	5.110E-01

Table 5.18: Results of SR of three algorithms

Functions	ABSEGA	ABSEGA2	ABSE-CMA-ES
F1(1D)	1.000E+00	1.000E+00	1.000E+00
F2(1D)	1.000E+00	1.000E+00	1.000E+00
F3(1D)	1.000E+00	1.000E+00	1.000E+00
F4(2D)	1.000E+00	1.000E+00	1.000E+00
F5(2D)	1.000E+00	1.000E+00	1.000E+00
F6(2D)	0.000E+00	0.000E+00	1.000E+00
F7(2D)	1.818E-01	0.000E+00	6.452E-01
F6(3D)	0.000E+00	0.000E+00	0.000E+00
F7(3D)	0.000E+00	0.000E+00	0.000E+00
F8(2D)	1.000E+00	1.000E+00	1.000E+00
F9(2D)	0.000E+00	0.000E+00	1.000E+00
F10(2D)	0.000E+00	0.000E+00	0.000E+00
F11(2D)	0.000E+00	0.000E+00	7.419E-01
F11(3D)	0.000E+00	0.000E+00	0.000E+00
F12(3D)	0.000E+00	0.000E+00	0.000E+00
F11(5D)	0.000E+00	0.000E+00	0.000E+00
F12(5D)	0.000E+00	0.000E+00	0.000E+00
F11(10D)	0.000E+00	0.000E+00	0.000E+00
F12(10D)	0.000E+00	0.000E+00	0.000E+00
F12(20D)	0.000E+00	0.000E+00	0.000E+00

Table 5.19: Results of Friedman's Test of three algorithms

Component	Ranking
ABSEGA	2.400E+00
ABSEGA2	2.267E+00
ABSE-CMA-ES	1.333E+00
Iman and Davenport	7.141E+00
P-value	3.119E-03

Table 5.20: Holm's Test for $\alpha = 0.05$ of three algorithms

i	algorithms	$z = (R_0 - R_i)/SE$	p	Holm
3	ABSE-CMA-ES vs ABSEGA	2.921E+00	3.487E-03	0.017
2	ABSE-CMA-ES vs ABSEGA2	2.556E+00	1.059E-02	0.025
1	ABSEGA vs ABSEGA2	3.651E-01	7.150E-01	0.05

Table 5.21: Contrast estimation of three algorithms

	ABSEGA	ABSEGA2	ABSE-CMA-ES
ABSEGA	0.000E+00	-5.233E-03	-4.948E-01
ABSEGA2	5.233E-03	0.000E+00	-4.895E-01
ABSE-CMA-ES	4.948E-01	4.895E-01	0.000E+00

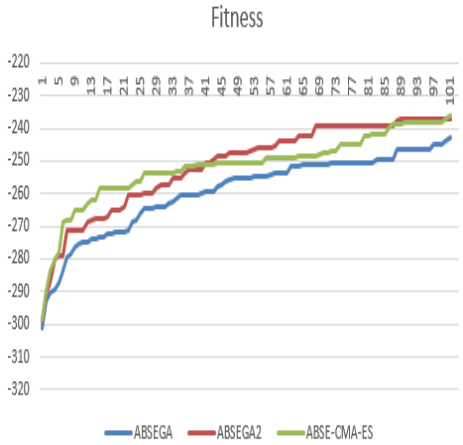


Figure 5.2: Fitness of three algorithms on the ball catching task

fitness and diversity as ABSEGA2. Both of them are better than ABSEGA. The diversity on three levels is shown in Table. 5.22. ABSE-CMA-ES is worse than ABSEGA2 on the -280 level, but better than it on the -320 level. They have similar diversity on -380 level.

To conclude, ABSE-CMA-ES is better than rest two algorithms. ABSEGA2 is slight better than ABSEGA.

Table 5.22: Diversity of three algorithms at different levels

Levels	-280	-320	-380
ABSEGA	1.91	2.82	3.45
ABSEGA2	3.363	4.45	5.63
ABSE-CMA-ES	2.81	5.09	5.54

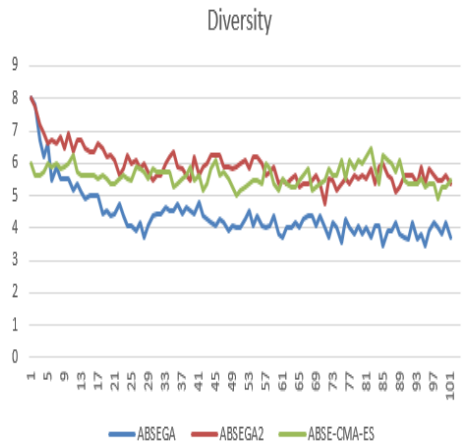


Figure 5.3: Diversity of three algorithms on the ball catching task

5.4.4 Visualization

To make an easy understanding of ABSE, snapshots of the algorithm on F7(2d) and F8(2d) are showed on Fig.5.4. The background of those snapshots are contour lines of the function; the green and red points are individuals and detected seeds respectively; the circle around a seed is the estimated ABS. Those snapshots are taken on every 100 generations.

Those snapshots show how the algorithm works. At generation 0 (Fig.5.4.a), only a few seeds are detected. Their positions are far from peaks, and their ABSs are not accurate. In the process of running, more seeds are detected. Both their positions and ABSs become more accurate. Especially, in Fig.5.4.b.F8(2d) here is a notable mistake on the middle-top of the graph, in which the ABS of a seed is too big. This mistake is fixed afterwards, so the ABS becomes small on Fig.5.4.c.F8(2d).

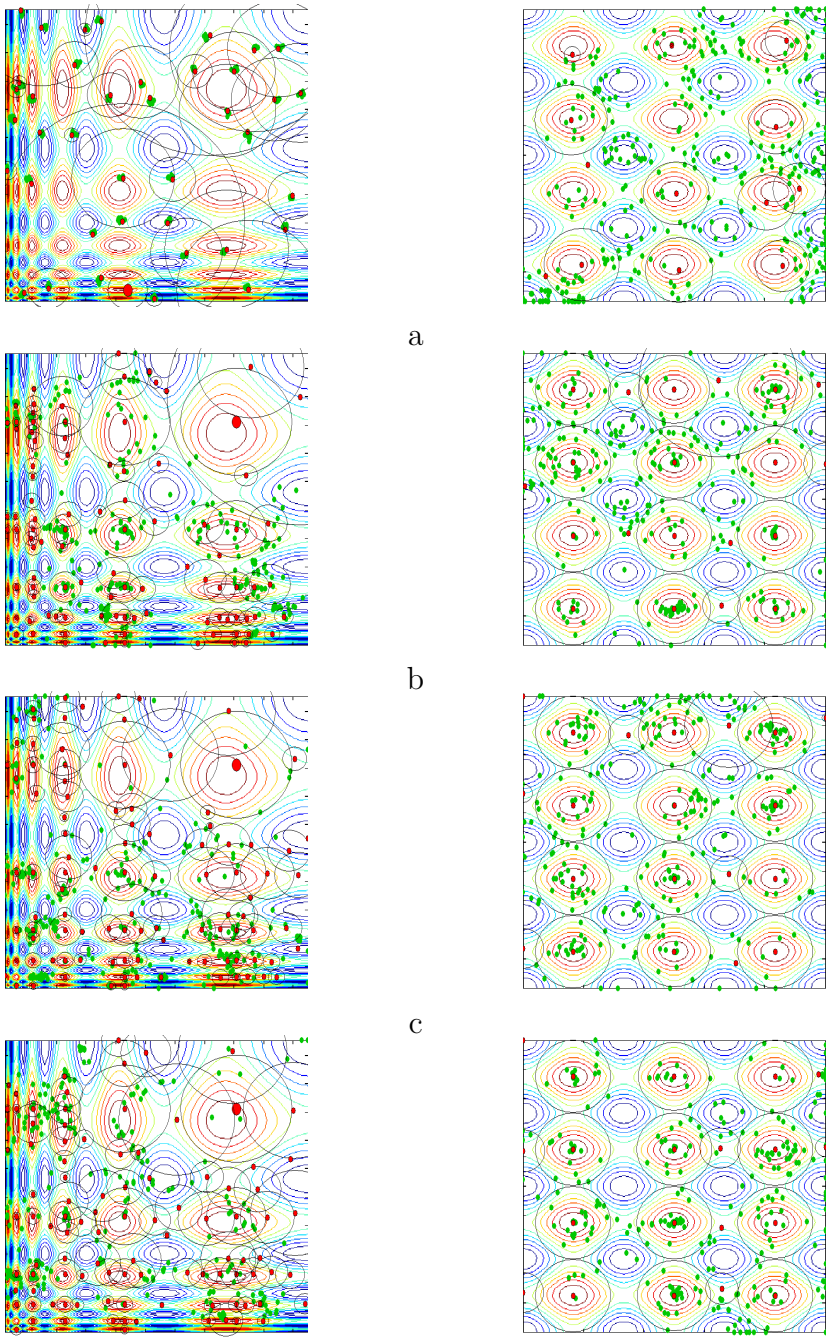


Figure 5.4: F72d (left) and F82d (right) on 0,100,...300 generation

5.5 Conclusion

ABSE-CMA-ES is a new niching evolutionary algorithm which combines CMA-ES, Attraction Basin Sphere Estimation (ABSE) and Seed Importance Calculations (SIC2).

Experiments were performed to examine the performance of ABSE-CMA-ES. The algorithm was compared with the best 5 niching methods in CEC 2013 Niching Methods Competition. The results showed that the proposed algorithm is significant better than the dADE/nrand/1, DE/nrand/2, CMA-ES with a simple archive and NVMO. It is also better than NEA2, but not in a significant way. The results suggested that there is a relation among the parameters, the desired number of solutions of the problem and the performance, but not in a significant way. The results also showed that ABSE is very important to obtain a high performance. SIC2 can improve the performance, but not in a significant way. Their combination can obtain the best performance. ABSE-CMA-ES is better than ABSEGA and ABSEGA2. However, ABSEGA2 is slight better than ABSEGA.

Chapter 6

Conclusion

In Chapter 1, I introduced the background and purpose of this thesis.

In Chapter 2, I proposed a niching method named as Attraction Basin Sphere Estimation (ABSE). This method can collect niching information about the landscape. Some examples were given to describe how ABSE works.

In Chapter 3, I combined ABSE and standard genetic algorithms to create a niching genetic algorithm: Attraction Basin Sphere Estimation Genetic Algorithm (ABSEGA). It identifies optimal-like individuals (seeds) from the population, and uses ABSE to calculate niching parameters. The experiments were performed on benchmark tests. I compared the proposed method with another adaptive niching method: Topological Species Conservation (TSC). The results indicated that ABSEGA has a similar ability to solve

multimodal optimization problems as TSC. Comparing the computational cost, ABSEGA was found to perform more efficiently than TSC. Examining the randomly sampled parameter settings, the results were found to be insensitive to different parameter settings on most test functions.

In Chapter 4, I improved ABSEGA for Neuroevolution problems. I proposed a method, SIC, to calculate the Importance of optimal solutions. Importance measures the possibility of an optimal solution to be a global optimal solution, so the new algorithm, ABSEGA2, can find multiple global optimal solutions. I first examined ABSEGA2 with some state-of-the-art algorithms including dADE, DE, CMA-ES with a simle archive, and NEA2 on benchmark tests. Second, I examined this method in a robotic arm problem, in which I evolved an artificial neural network (ANN) to control the arm to catch balls. I compared ABSEGA2 with a radius-based method Dynamic Fitness Sharing (DFS), the Standard Genetic Algorithm and those state-of-the-art algorithms. The results showed that ABSEGA2 has a high ability to escape local optimal and find relatively better solutions. The experiments also showed that the calculation of Importance plays an important role on keeping diversity on complex tasks.

In Chapter 5, I combined ABSE and CMA-ES, a very powerful local optimization method. The proposed niching method is named as Attraction Basin Sphere Estimation CMA-ES (ABSE-CMA-ES). The algorithm was

compared with the best 5 niching methods in CEC 2013 Niching Methods Competition. The results showed that the proposed algorithm is significant better than the dADE/nrand/1, DE/nrand/2, CMA-ES with a simple archive and NVMO. It is also better than NEA2, but not in a significant way. I compared the three proposed methods on both benchmark tests and the ball catching task. The results showed that ABSE-CMA-ES is better than ABSEGA and ABSEGA2. However, ABSEGA2 is only slight better than ABSEGA.

Finally, I gives some directions toward the future work.

1. Propose an algorithm that can solve Multiobjective Optimization and Multimodal Optimization in the same time.

2. Improve ABSE to provide a common solution for structured problems where it is difficult to define interior points between two solutions.

3. Adaptively adjust more parameters.

Acknowledgment

The author expresses to thank Professor Masahito Yamamoto of Hokkaido University for constant patient instruction and guidance from the inception to the completion of the present research.

Deep thanks are also due to Professor Masahito Kurihara, Professor Tetsuo Ono, Professor Keiji Suzuki, Associate Professor Hiroyuki Iizuka of Hokkaido University.

He would like to thank Associate Professor Ikuo Suzuki of Kitami Institute of Technology and Professor Masashi Furukawa of Hokkaido Information University for profitable discussion and suggestions. Many pieces of affectionate advice and discussion given by them were very useful to improve this research.

He would like to thank all students of the office, Autonomous Systems Engineering Laboratory, Complex Systems Engineering. The conversations between them encouraged him.

He would like to gratefully thank the financial support from China Scholarship Council.

Finally, the author would like to thank to his family; his father Hou Dongmin, his mother Xu Liping and his wife Wang Jing. Without his family, he would not have continued his research.

Bibliography

[1] Petrowski A. A clearing procedure as a niching method for genetic algorithms. In *In Proceedings of IEEE International Conference on Evolutionary Computation*, pages 798–803. IEEE, May 1996.

[2] Thomas Back and Hans Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[3] A. Blanco. A real-coded genetic algorithm for training recurrent neural networks. *Neural Networks*, 14(1):93–105, 2001.

[4] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. CAMBRIDGE UNIVERSITY PRESS, 2004.

[5] Antonio Della Cioppa, Claudio De Stefano, and Angelo Marcelli. Where are the niches? dynamic fitness sharing. *IEEE Trans. Evolutionary Computation*, 11(4):453–465, Aug 2007.

[6] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE Trans. Evolutionary Computation*, 15(1):4–31, 2011.

[7] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

[8] Michael G. Epitropakis, Xiaodong Li, and Edmund K. Burke. A dynamic archive niching differential evolution algorithm for multimodal optimization. In *In Proceedings of Congress of Evolutionary Computation(CEC 2013)*, pages 79–86. IEEE Press, june 2013.

[9] Michael G. Epitropakis, Vassilis P. Plagianakos, and Michael N. Vrahatis. Finding multiple global optima exploiting differential evolutions niching capability. In *In 2011 IEEE Symposium on Differential Evolution(SDE)*, pages 1–8. IEEE Press, April 2011.

[10] D. Floreano, P. Durr, and C. Mattiussi. Neuroevolution: From architectures to learning. *Evolutionary Intelligence.*, 1(1):4762, Mar 2008.

[11] Nikolaus Hansen. The CMA evolution strategy: A comparing review towards a new evolutionary computation. In *Towards a New Evolutionary*

Computation, volume 192 of *Studies in Fuzziness and Soft Computing*, chapter 4, pages 75–102. Springer Berlin / Heidelberg, 2006.

[12] Georges Harik. Finding multimodal solutions using restricted tournament selection. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 24–31. Morgan Kaufmann, 1995.

[13] John H. Holland. *Adaptation in natural and artificial systems.* MIT Press, 1992.

[14] De Jong and Kenneth Alan. *An analysis of the behavior of a class of genetic adaptative systems.* PhD thesis, University. of Michigan, 1975.

[15] Rasmus k. Ursem. Multinational evolutionary algorithms. In *In Congress of Evolutionary Computation (CEC 1999)*, pages 1633–1640. IEEE Press, Jul 1999.

[16] Rasmus k. Ursem. Multinational gas: Multimodal optimization techniques in dynamic environments. In *In Genetic and Evolutionary Computation Conference (GECCO 2000)*, pages 19–26. Morgan Kaufmann, Jul 2000.

[17] J. Kennedy and R. Eberhart. Particle swarm optimization. *Neural Networks, 1995. Proceedings., IEEE International Conference on*, 4:1942–1948 vol.4, 1995.

[18] Xiaodong Li, Andries Engelbrecht, and Michael G. Epitropakis. Benchmark functions for cec'2013 special session and competition on niching methods for multimodal function optimization. Technical report, Evolutionary Computation and Machine Learning Group, RMIT University, Australia, 2013.

[19] Daniel Molina, Amilkar Puris, Rafael Bello, and Francisco Herrera. Variable mesh optimization for the 2013 cec special session niching methods for multimodal optimization. In *In Proceedings of Congress of Evolutionary Computation(CEC 2013)*, pages 87–94. IEEE Press, june 2013.

[20] Hansen N and Ostermeier. A completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation.*, 9(2):159–195, Jun 2001.

[21] Costas Neocleous and Christos Schizas. Artificial neural network learning: A comparative review. In *In Proceedings of Second Hellenic Conference on AI*, pages 300–313. Springer Berlin Heidelberg, April 2002.

[22] Mike Preuss. Niching the cma-es via nearest-better clustering. In *In Proceedings of Genetic and Evolutionary Computation Conference Companion (GECCO 2010 Companion)*, pages 1711–1718. ACM Press, july 2010.

[23] Mike Preuss, Lutz Schnemann, and Michael Emmerich. Counteracting genetic drift and disruptive recombination in (μ,λ)-ea on multimodal fitness landscapes. In *In Proceedings of Genetic and Evolutionary Computation Conference(GECCO 2005)*, pages 865–872. ACM Press, june 2005.

[24] Amilkar Puris, Rafael Bello, Daniel Molina, and Francisco Herrera. Variable mesh optimization for continuous optimization problems. *Soft Computing*, 16(3):511–525, 2012.

[25] Bruno Sareni and Laurent Krähenbühl. Fitness sharing and niching methods revisited. *IEEE Trans. Evolutionary Computation.*, 2(3):97–106, Sep 1998.

[26] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *The 1998 IEEE International Conference on Evolutionary Computation Proceedings*, pages 69–73, 1998.

[27] Ofer M. Shir and Thomas Back. Dynamic niching in evolution strategies with covariance matrix adaptation. In *In Proceedings of Congress of Evolutionary Computation (CEC 2005)*, pages 2584–2591. IEEE Press, Sept 2005.

[28] Ofer M. Shir, Michael Emmerich, and Thomas Back. Adaptive niche radii and niche shapes approaches for niching with the cma-es. *Evolutionary Computation*, 18(1):97–126, Feb 2010.

[29] Catalin Stoean, Mike Preuss, D. Dumitrescu, and Ruxandra Stoean. Multimodal optimization by means of a topological species conservation algorithm. *IEEE Trans. Evolutionary Computation*, 14(6):842–864, Dec 2010.

[30] Catalin Stoean, Mike Preuss, Ruxandra Stoean, and D. Dumitrescu. Disburdening the species conservation evolutionary algorithm of arguing with radii. In *In Genetic and Evolutionary Computation Conference (GECCO 2007)*, pages 1420–1427. ACM New York, Jul 2007.

[31] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger, and S. Tiwari. *Problem definition and evaluation criteria for the CEC 2005 special session on real-parameter optimization.* Nanyang Technological University, Singapore, 2005.

[32] Mahfoud Samir W. *Niching methods for genetic algorithms.* PhD thesis, University of Illinois at Urbana-Champaign, 1995.

[33] Yu Xinjie and Gen Mitsuo. Multimodal optimization. *Introduction to Evolutionary Algorithms*, 0(2):165–191, 2010.

[34] Zhuoran Xu, Hiroyuki Iizuka, and Masahito Yamamoto. Attraction basin sphere estimating genetic algorithm for neuroevolution problems. *Artificial Life and Robotics*, 19(0):0, 2014.

[35] Zhuoran Xu, Mikko Polojarvi, Masahito Yamamoto, and Masashi Furukawa. Attraction basin estimating ga: An adaptive and efcient technique for multimodal optimization. In *In Proceedings of Congress of Evolutionary Computation (CEC 2013).*, pages 333 – 340. IEEE Press, Jue 2013.

[36] Zhuoran Xu, Mikko Polojarvi, Masahito Yamamoto, and Masashi Furukawa. An attraction basin estimating genetic algorithm for multimodal optimization. In *In Proceedings of Genetic and Evolutionary Computation Conference Companion (GECCO 2013 Companion)*, pages 131–132. ACM Press, Jul 2013.

[37] Zhuoran Xu and Masahito Yamamoto. Abega-2: Improved attraction basin estimating genetic algorithm for high dimensional space. In *In Proceedings of 19th International Symposium on Artificial Life and Robotics AROB*, pages 275–279, 2014.