B.TECH. PROJECT REPORT

On

Multi-Strategy Differential Evolution for Multimodal Optimization Problems

BY

Prathamesh Naik, 160001037
&
Arjun Srivastava, 160001007

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE

December 2019

Multi-Strategy Differential Evolution for Multimodal Optimization Problems

PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degrees

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

Prathamesh Naik, 160001037
&
Arjun Srivastava, 160001007,

Discipline of Computer Science and Engineering,

Indian Institute of Technology, Indore

Guided by:

Dr. Aruna Tiwari,

Associate Professor,

Computer Science and Engineering,

IIT Indore

INDIAN INSTITUTE OF TECHNOLOGY INDORE
December 2019

Declaration of Authorship

We hereby declare that the project entitled **"Multi-Strategy Differential Evolution for Multimodal Optimization Problems "** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in Computer Science and Engineering completed under the supervision of **Dr. Aruna Tiwari, Associate Professor, Computer Science and Engineering, IIT Indore** is an authentic work.
Further, we declare that we have not submitted this work for the award of any other degree elsewhere.

Signatures (with date):

Prathamesh Naik *Arjun Srivastava*

Certificate

This is to certify that the thesis entitled, *"Multi-Strategy Differential Evolution for Multimodal Optimization Problems "* and submitted by Prathamesh Naik (ID No 160001037) and Arjun Srivastava (ID No 160001007) in partial fulfillment of the requirements of CS 493 B.Tech Project embodies the work done by them under my supervision.

Supervisor

Dr.ARUNA TIWARI
Associate Professor,
Indian Institute of Technology
Indore
Date:

“If I have seen further than others, it is by standing upon the shoulders of giants. ”

Isaac Newton

INDIAN INSTITUTE OF TECHNOLOGY INDORE

Abstract

Department of Computer Science and Engineering

Bachelor of Technology

Multi-Strategy Differential Evolution for Multimodal Optimization Problems

Many real world problems require finding multiple solutions. Multiple optimal solutions help in providing a better understanding of complex systems, as they reveal the underlying patterns of such systems. They're useful as candidate solutions to reduce the search space for solving real life problems, as they allow us to make focused trade offs within this constrained set, rather than needing to consider the full search space. They're necessary when using imperfect simulators, as finding one optimal solution isn't enough because it may not be suitable in real life. We need to identify multiple optimal solutions for decision makers to choose from. But searching for multiple optimal solutions simultaneously is known to be much more challenging than traditional uni-modal optimization.

We present an exhaustive literature review, surveying the best techniques to solve the problem of multi-modal optimisation. In particular we study and implement **DSDE**, **DSDE-C** *(Dual-Strategy Differential Evolution With Affinity Propagation Clustering)* & **FBK-DE** *(formulation, balance, and keypoint - a Differential Evolution algorithm using Nearest-Better Clustering)*. By studying and implementing the state of the art algorithms in this field, we also replicate their corresponding results.

We go one step further by proposing a novel algorithm which combines currently known techniques such as archival from DSDE and Nearest Better Clustering with Min-size from FBK-DE, and adds a Multi Armed Bandit strategy for generating new individuals along with other novelties such as the use of a softmax temperature strategy in clustering.

We perform extensive experimention for hyper-parameter optimization as well as for choosing the best combination of strategies. As part of our work we also create a framework to parallelize the testing on such a big dataset. It provides a linear speedup which helped us reduce our testing time by over 30 times.

Based on the experimentation, we present two different versions of our algorithm which show different characteristics on different kinds of problem types. In particular, the first version is suitable for low dimensional problems while the latter one is meant for higher dimensional problems.

We achieve an improved performance on CEC2013 which is the benchmark dataset of this field. We improve scores by 9% and reduce error rate by 22%.

Acknowledgements

We would like to thank our B.Tech Project supervisor **Dr. Aruna Tiwari** for her guidance and constant support in structuring the project and her valuable feedback throughout the course of this project. Her overseeing the project meant there was a lot that we learnt while working on it.

We are grateful to **Ms. Suchitra Agrawal** who provided valuable guidance in shaping the project. We thank her for her time and effort.

Most importantly, we are thankful for each other's camaraderie, without which writing the thesis would have taken much longer. Also, we are thankful to our other friends who were a constant source of both motivation and light hearted humour.

We are really grateful to the Institute for the opportunity to be exposed to systemic research, especially, Dr. Aruna Tiwari's lab for providing the necessary hardware utilities to complete the project.

Lastly, we offer our sincere thanks to everyone who helped us to complete this project, whose name we might have forgotten to mention.

Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

BSS	**B**est **S**ub-Specie **A**lgorithm
CEC	**C**ongress **O**n **E**volutionary **C**omputation
DSDE	**D**ual **S**trategy **D**ifferential **E**volution
FBK-DE	**F**ormulation **B**alance **K**eypoint **D**ifferential **E**volution
M	**M**ulti **A**rmed **B**andit **G**enerate
MAR	**M**inimum **A**ccpeptable **R**ate
MAS	**M**aximum **A**cceptable **S**ize
MI	**M**ulti **A**rmed **B**andit **I**terative **G**enerate
MIR	**M**ulti **A**rmed **B**andit **I**terative **R**epeated **G**enerate
MMOP	**M**ulti **M**odal **O**ptimisation **P**roblem
MR	**M**ulti **A**rmed **B**andit **R**epeated **G**enerate
NBC	**N**earest **B**etter **C**lustering
PR	**P**eak **R**atio
SR	**S**uccess **R**atio

Chapter 1

Introduction

1.1 Background

The aim of this project is threefold:

- Studying and implementing the current state of the art evolutionary algorithms for solving multimodal optimization problems (MMOP: problems having multiple optimal solutions)
- Designing a novel evolutionary algorithm to solve multimodal optimization problems
- Testing the new approach on standard benchmark sets and analysing the results

Searching for multiple optimal solutions simultaneously is known to be much more challenging than traditional unimodal optimization.

- Unimodal optimization

$$\text{Find any} X_0 : f(X_0) >= f(x)\ \forall x \in Domain$$

- Multimodal optimization

$$\text{Find all } X_0 : f(X_0) >= f(x)\ \forall x \in Domain$$

Solving MMOPs requires carefully balancing exploration and exploitation. Exploration is required to maintain population diversity so as to locate as many global optima as possible, while exploitation is needed to increase the accuracy of the solutions found.

In recent years, efforts have been made to extend evolutionary algorithms to MMOPs. The key idea has been to partition the population and apply evolutionary techniques within the subpopulations, with the goal that eventually each subpopulation will find a different optima. In this project, we will analyse and implement certain well known approaches and design a new algorithm to try to improve the state of the art.

1.2 Why do we need to find multiple optimas?

Many real world problems admit multiple optimal solutions. For many such problems, it is often preferable (or even required) to find multiple optima. Some of the reasons are noted below.

- **Better understanding**: For better understanding of complex systems by finding all possible optimal configurations. Finding one optimal solution for complex systems is not very enlightening. By finding all possible optimal configurations, we gain a much better understanding of its underlying principles.

  Eg. Pareto optimization in economics
  Affordability and taste determine how likely customers are to go to a restaurant. We can setup the problem formally, find multiple optimal solutions and accurately determine the tradeoff between affordability and taste.

- **Reducing search space**: Finding candidate solutions to reduce the search space to solve real world problems. By having all of the potentially optimal solutions, we can treat them as candidate solutions and we can make focused trade offs within this constrained set, rather than needing to consider the full search space.

  Eg. Multiple solutions of the travelling salesman problem
  A person can choose among various solutions (i.e. paths) based on traffic conditions, weather predictions, transport availability etc.

- **Imperfect simulators**: Simulators used in fields such as robotics aren't perfect; finding one optimal solution isn't enough as it may not be suitable in real life. We need to identify multiple optimal solutions for decision makers to choose from.

  Eg. Finding good policies using robotic simulators that actually work in the real world.
  If we're trying to find an optimal policy for a robotic hand, it's very slow and expensive to test it in the real world. So instead we would use a robotic simulator. But since the simulators aren't perfect, the policy we find may not be suitable in the real world. So we would want to find multiple solutions and then we could choose from them.

1.3 Objectives

- Studying evolutionary algorithm concepts

  We'll study differential evolution, population initialization, mutation operators, crossover operators, selection operators as well as niching (or clustering) techniques.

- Studying and implementing the two algorithms for *Dual-Strategy Differential Evolution With Affinity Propagation Clustering* (**DSDE** and **DSDE-C**)[1]

  We'll study & implement these two algorithms as they introduce novel concepts like dual-strategy evolution & archival.

- Studying and implementing **FBK-DE** (formulation, balance, and keypoint - a *Differential Evolution algorithm using Nearest-Better Clustering*)[2]

  We'll study & implement this algorithm as it is the current state of the art evolutionary algorithm for multi-modal optimization problems. It also introduces a new clustering method (Nearest-Better-Minsize clustering).

- Understanding and using the standard *CEC'2013 benchmark dataset* [3]

  We'll test our approaches using this benchmark suit, as it will aid in comparing the performance of our new design proposals with each other and with the current state of the art algorithms.

- Designing and implementing an evolutionary algorithm for solving multi-modal optimization problems

  We'll combine the concepts learned from the three algorithms that we've implemented and propose novel strategies for clustering, generation & archival to create a new algorithm.

- Experimentation followed by analysis of results

  Experimentation will be used for hyper-parameter optimization as well as for choosing the best strategies.

- Creating accessible resources to explain the new algorithm

1.4 Organization of this thesis

In this chapter, we introduced the problem, explained the motivation behind multi-modal optimization and specified the objectives of our project. The rest of the report is organized as follows.
Chapter 2 provides the literature review to bring the reader up to speed to the current state of art in solving multimodal optimization problems and also helps to provide a foundation for contrasting our Final algorithm. Chapter 3 describes design proposals to help improve the state of the art and also the design of our final proposed algorithm. We provide the high level overview of our algorithm with all necessary details for the reader's benefit. Chapter 4 elaborates upon the implementation details in replicating previous work (DSDE, DSDE-C and FBK-DE) and coding our proposed algorithm along with the related software engineering considerations. Chapter 5 provides the necessary details related to evaluating different candidate solutions. In Chapter 6, we present all relevant data and results and compare the performance of our algorithm relative the the previous state of the art. Finally we conclude in Chapter 7 providing a summary and potential future directions of extending our work.

Chapter 2

Literature Review

The following chapter discusses literature pertaining to previously known methods of multi-modal optimization using evolutionary techniques.

2.1 Standard evolutionary approach to solve MMOPs

To start with, we have a fitness function which we want to maximise. We randomly initialise a population of **N** individuals (an individual is just a point in function domain) within the given bounds of the function domain. Then we evaluate the population, i.e. find the fitness of all the individuals by evaluating the fitness function on all of them.

Using the fitness and spatial information (coordinates corresponding to individuals), we partition the population using some method such as speciation, crowding, etc. This process is called clustering. The resulting sub-populations are also referred to as species or niches.

Next we apply evolutionary techniques (including different kinds of mutation and crossover operators) inside each of the sub-populations. We evaluate the new generation of individuals (i.e., calculate their fitness).

Finally we select individuals from the parents as well as children that are good enough to enter the next generation. These selected individuals form the new population and the process is repeated until we reach the termination criteria (which is usually the maximum number of fitness function evaluations allowed).

The flowchart for the whole procedure is given on the next page.

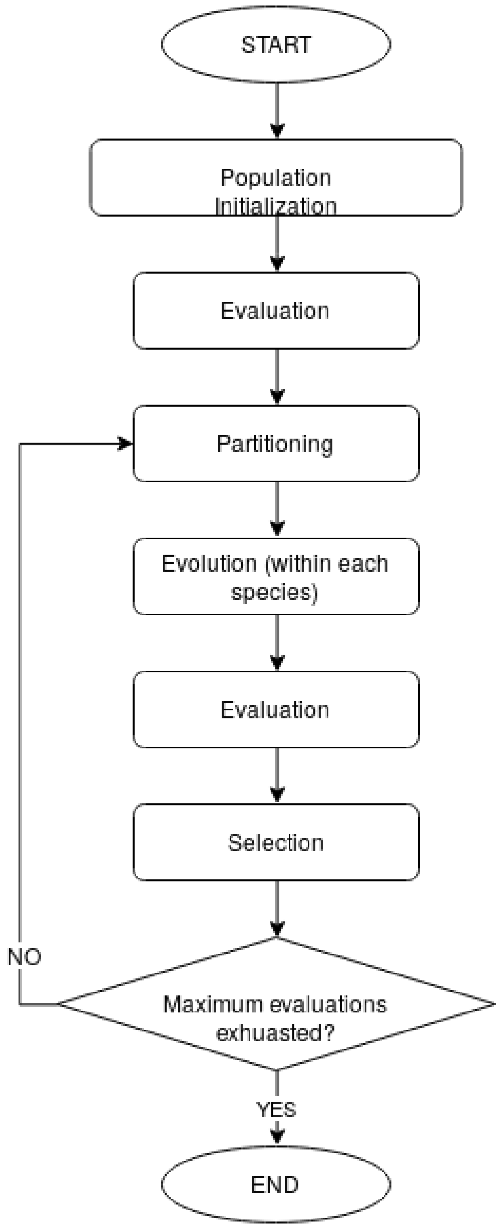
START
Population Initialization
Evaluation
Partitioning
Evolution (within each species)
Evaluation
Selection
Maximum evaluations exhuasted?
NO
YES
END

FIGURE 2.1: Standard evolutionary approach to solve MMOPs

2.1.1 Initialization

The initial population is randomly initialised as follows.

$$x_i^j(0) = rand(0,1) * (x_{max}^j - x_{min}^j) + x_{min}^j \tag{2.1}$$

Here $x_i^j(g)$ represents the jth dimension value of the ith individual x_i at gth generation. x_{max} and x_{min} are the upper and lower bounds of the jth dimension. i varies from 1 to NP (the size of the population) and j varies from 1 to D (dimensions of the problem). $rand(0,1)$ represents a random value drawn from a uniform distribution from 0 to 1.

2.1.2 Partitioning (niching or clustering)

There are various methods for partitioning that we studied and implemented. They are described below.

Speciation clustering

Algorithm 1 Speciation clustering (from [1])

```
Begin
1.  Sort the population from better to worse according to fitness value;
2.  For i=1 to N/M //N is the population size, M is the cluster size
3.      The best individual X is set as the species seed;
4.      The X and its nearest M − 1 individuals form a new species;
5.      Remove the M individuals from the population;
6.  End of For
End
```

In speciation, the population is divided into species (sub-populations). Each sub-population is formed by a species seed and its $(M-1)$ neighbors .
The best individual and it's nearest $(M-1)$ neighbours form a specie. Now they're removed from the population and this process is repeated until all species are formed.

Nearest better clustering (NBC)

Algorithm 2 Nearest better clustering (from [2])

```
 1: Sort the population $P$ by the fitness $F$ from the best to the
    worst;
 2: Calculate the distances between each pair of individuals;
 3: Create an empty tree $T$;
 4: for each $\boldsymbol{x}_i \in P$ do
 5:     Find the nearest better neighbor $\boldsymbol{x}_{i,nb}$;
 6:     Create an edge between $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i,nb}$
 7:     Add the edge to $T$;
 8: end for
 9: Calculate the mean distance $\mu_{dist}$ of all edges in $T$;
10: for each $e \in T$ do
11:     if $dist(e) > \varphi \times \mu_{dist}$ then
12:         Cut off the edge $e$ and record the seed;
13:     end if
14: end for
```

First, the population is sorted by the fitness in descending order. Then,the distance between each pair of individuals is calculated. Using these distances, each individual(except for the best one) finds its own nearest-better neighbor and an edge is created to connect them.
The nearest-better neighbor is called the leader individual for two nodes connected by an edge, while the individual itself is called the follower individual. After forming these connections, a spanning tree T is formed. After this process, the mean distance (i.e., length) μ_{dist} is calculated.
Next, all the edges whose distance is greater than $\phi * \mu_{dist}$ are cut off, where ϕ is called the scale factor of NBC.
At the end, the spanning tree T is divided into several subtrees. Each of them represents a species and the root of a subtree is regarded as the seed of the corresponding species.
As comapared to other clustering methods, NBC performs well. NBC also does not require prior knowledge such as the number of peaks. Further, NBC requires only one parameter ϕ to be set.
ϕ controls the number of species. When it is small,more species are obtained, while increasing ϕ results in fewer species. In general, ϕ is set to 2.0 [4]

Nearest better clustering with Minsize

Algorithm 3 Nearest better clustering with Minsize (from [2])

```
1:  Set $minsize$
2:  Construct the spanning tree $T$;
3:  Calculate the mean distance $\mu_{dist}$;
4:  Calculate the $follow$ vector;
5:  Sort the edges in $T$ from the longest to the shortest;
6:  for each $e \in T$ do
7:      if $dist(e) > \varphi \times \mu_{dist}$ then
8:          Set $e_f$ to the follower individual of $e$;
9:          Set $e_r$ to the root of the subtree containing $e_f$;
10:         if $follow(e_f) \geq minsize$ and
              $follow(e_r) - follow(e_f) \geq minsize$ then
11:             Cut off $e$;
12:             Set $e_l$ to the leader individual of $e$;
13:             for each $x$ on the path from $e_l$ to $e_r$ do
14:                 $follow(x) = follow(x) - follow(e_f)$;
15:             end for
16:         end if
17:     end if
18: end for
```

A spanning tree is constructed just like in NBC and the average distance μ_{dist} is calculated.
Next, we calculate the follow vector. The value of follow corresponding to an individual represents the number of nodes in the subtree rooted at that individual.
To calculate this, each value of follow is initialised to 1. After this, the edges are sorted in descending order according to the fitness values of the follower individuals. Then for each edge, the follow value of each follower individual is added to that of the corresponding leader individual.
Next, the edges in are sorted by their length in descending order. Then, edges satisfying the 2 conditions are cut off. The first is the condition of the standard NBC, that the distance exceeds the weighted average distance. The second condition is that the sizes of the two species after the cutoff must both greater than or equal to *minsize*. Finally, all species are returned.

2.2 Evolution

2.2.1 Mutation

Mutation operators are used to generate the mutant individuals. For an individual x, a mutation operator is used to convert it to a corresponding mutant individual v.

Example mutation operators:

- DE/rand/1
$$v_i(g) = x_{r3}(g) + F * (x_{r1}(g) - x_{r2}(g)) \tag{2.2}$$
- DE/best/1
$$v_i(g) = x_{best}(g) + F * (x_{r1}(g) - x_{r2}(g)) \tag{2.3}$$

Here, $r1$, $r2$, and $r3$ are indices of three random individuals from the sub-population. The parameter F (which is between 0 to 1) controls the effect of difference vectors. x_{best} is the individual with the best fitness in the current sub-population.

2.2.2 Crossover

Crossover operation exchanges some components from x_i and v_i to form a trial vector u_i , with each dimension determined as where j_{rand} is an integer uniformly selected from $1, 2, ..., D$ to make sure that at least one dimension of u_i comes from v_i.

$$u_{i,j} = \begin{cases} v_{i,j}, & \text{if } rand \leq CR \text{ or } j = j_{rand} \\ x_{i,j}, & \text{otherwise} \end{cases} \tag{2.4}$$

The crossover rate CR is another parameter, which decides the proportion of the trial vector inherited from the mutation vector v_i.

2.2.3 Selection

The selection operator is utilized to choose the best individual from $u_i(g)$ and $x_i(g)$ into the next generation shown as follows:

$$x_i = \begin{cases} u_i, & \text{if } f(u_i) \geq f(x_i) \\ x_i, & \text{otherwise} \end{cases} \tag{2.5}$$

2.3 Benchmark Functions for CEC'2013 Special Session and Competition on Niching Methods for Multimodal Function Optimization

In this technical report ([3]), the authors introduce a unifying framework for evaluating niching methods and multimodal optimization algorithms. Their goal was to create a standardised testing suite so that further advances in the area of multimodal optimization can be made with ease.
They have put together 20 benchmark test functions with different characteristics. The first 10 benchmark functions are simple, well known and widely used functions, largely based on earlier studies. The remaining benchmark functions are more complex and follow the paradigm of composition functions.

All the 20 multimodal test functions in the CEC 2013 benchmark set are maximization problems and have a given limit of maximum number of function evaluations (MaxFEs). We decided to choose this standard set of functions for experimentation so that we can readily compare the performance of various new approaches with existing algorithms.

The functions vary in complexity, dimensionality (1 to 20) and number of optima (upto 216). We used an implementation of this benchmark suite in Python. The benchmark includes methods to access the different functions and their properties. The benchmark also includes a method to check how many optima we found within a given accuracy level.

We used a Python implementation of the benchmark set (which can found at [5])

2.3.1 Plots (from [3])

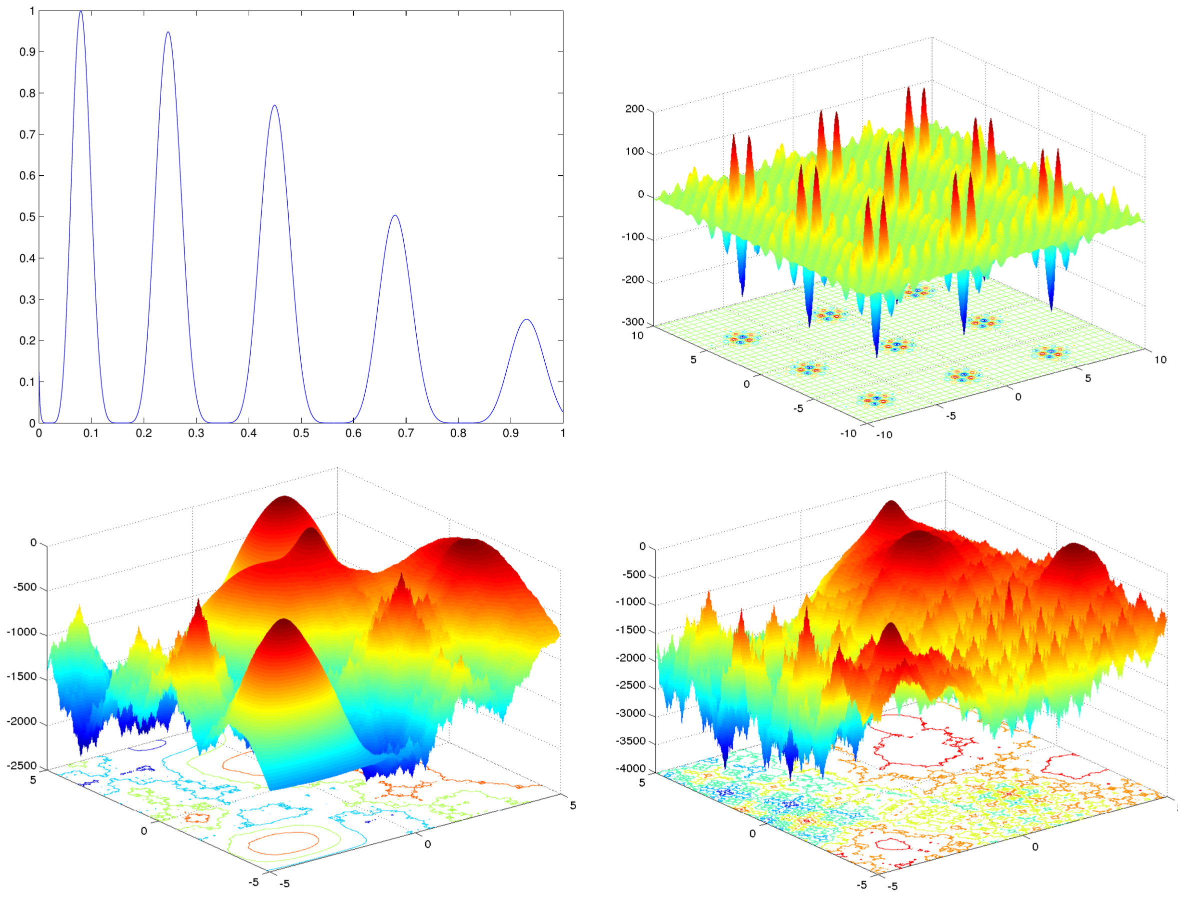

FIGURE 2.2: Examples of functions from the benchmark dataset

2.3.2 Metrics

The following two metrics are used to measure performance on any problem from this benchmark set:

- **Peak ratio (PR)**: The fraction of optimas found for a particular accuracy level (averaged over multiple runs)

  $$PR = \frac{\sum_{run=1}^{NR} NPF_i}{NKP * NR} \tag{2.6}$$

  NPF_i (number of peaks found) denotes the number of global optima found at the end of the i-th run, NKP (number of known peaks) the number of known global optima and NR the number of runs.

- **Success ratio (SR)**: The fraction of times we find all optimas

  $$SR = \frac{NSR}{NR} \tag{2.7}$$

  NSR denotes the number of successful runs.

2.4 Dual-Strategy Differential Evolution With Affinity Propagation Clustering for Multimodal Optimization Problems

2.4.1 DSDE

Algorithm 4 DSDE (from [1])

```
Begin
1.  Randomly initialize the population;
2.  Randomly generate an integer as the cluster size $M$:
3.  Partition the population into several species
4.  For each species
5.      For each individual $x_i$ in the current species
6.          If $x_i$ is a superior individual
7.              Using DE/lbest/1 strategy for $x_i$;
8.          Else
9.              Using DE/current-to-rand/1 strategy for $x_i$;
10.         End of If
11.     End of For
12. End of For
13. Perform adaptive selection mechanism
14. For each individual $x_i$
15.     If $sc_i \geq T$
16.         Move $x_i$ and its neighbors whose fitness values are worse
            than $x_i$ to archive and reinitialize these individuals;
17.     End of If
18. End of For
19. Repeat Steps 2 to 18 until the termination criterion is satisfied.
End
```

2.4.2 DSDE-C (selection strategy)

Algorithm 5 DSDE-C selection (from [1])

```
For each offspring $u_i$
    Compare $u_i$ with the nearest parental individual
    and replace it if $u_i$ has a better fitness value;
End of For
```

This paper introduces 2 algorithms: **DSDE** & **DSDE-C** (dual strategy differential evolution) which have certain novel characteristics. They use a dual-strategy mutation scheme to balance exploration and exploitation in generating offspring. This involves dividing each subpopulation into superior and inferior parts based on the fitness of

individuals, and then using 2 different mutation strategies according to whether an individual is superior or inferior.
In the above pseudocode for DSDE, the partitioning algorithm used is speciation clustering (described earlier: Algorithm 1).

2.4.3 Adaptive selection

Algorithm 6 Adaptive selection mechanism (from [1])

```
Begin
1.   Execute APC on the combined population;
2.   Calculate the choosing probability $P_i$ for each subpopulation;
3.   $N_0 = 0$; $i = 1$;
4.   **While** $N_0 < N$
5.       **If** Subpopulation $i$ is not empty
6.           **If** $rand < P_i$
7.               Remove the best individual $X$ in subpopulation $i$ and
                 add $X$ to the next generation;
8.               $N_0 = N_0 + 1$;
9.           **End of If**
10.      **End of If**
11.      $i = i + 1$;
12.      **If** $i > n$ ($n$ is the number of subpopulations identified by APC)
13.          $i = 1$;
14.      **End of If**
15.  **End of While**
End
```

The algorithms use an adaptive selection mechanism based on APC (Affinity Propagation Clustering) to choose diverse individuals from different optimal regions for locating as many peaks as possible. The choosing probability for each subpopulation is calculated as follows:

$$P_i = \frac{f_i - f_{min} + \phi}{f_{max} - f_{min} + \phi} \tag{2.8}$$

Here, f_i is the fitness of the species seed in the ith subpopulation, f_{max} and f_{min} are the maximum and minimum fitness among all the species seeds. To avoid division by zero, ϕ is set as 0.0001.

Lastly, the algorithms include an archival technique to detect and preserve stagnated and converged individuals. These individuals are stored in the archive to preserve the found promising solutions and are reinitialized for exploring new areas.

2.5 Differential Evolution for Multimodal Optimization With Species by Nearest-Better Clustering

This paper introduces the **FBK-DE** (formulation, balance, and keypoint - differential evolution) algorithm, which is the current state of the art in evolutionary multimodal optimization techniques.

Algorithm 7 FBK-DE (from [2])

```
1: Set the population size $NP$;
2: Randomly generate the initial population $P(0)$;
3: Evaluate the initial population $P(0)$;
4: $g = 0$;
5: while the termination condition is not satisfied do
6:     Obtain the species
7:     Obtain the size of the species
8:     for each species $s_i$ do
9:         for i=1 to $min(sNP_i, nums(i))$ do
10:            Generate $v_i(g)$
11:            Generate $u_i(g)$
12:            Put the better into $P(g+1)$
13:        end for
14:        if $nums(i) > sNP_i$ then
15:            Generate new individuals
16:            Insert the new one into $P(g+1)$;
17:        end if
18:    end for
19:    $g = g + 1$;
20: end while
```

First, it uses nearest-better clustering (NBC) with minsize (Algorithm 3 to divide the population into multiple species with minimum size limitations.

Second, to avoid placing too many individuals into one species, a species balance strategy (Algorithm 8) is used to adjust the size of each species.

Third, two keypoint-based mutation operators named DE/keypoint/1 and DE/keypoint/2 are proposed to evolve each species together with traditional mutation operators (Algorithm 9). Fourth, new individuals are generated using Equation 3.2.

2.5.1 Balance

Algorithm 8 Balance (from [2])

```
$rest = 0$;
$\mu_{avg} = mean(nums)$;
$\mu_{\lambda} = round(\lambda \cdot \mu_{avg})$;
$s = \varnothing$
for $i = 1$ to $len(nums)$ do
    if $nums(i) > \mu_{\lambda}$ then
        $rest = rest + nums(i) - \mu_{\lambda}$;
        $nums(i) = \mu_{\lambda}$;
    else if $nums(i) < \mu_{avg}$ then
        $s = s \cup \{i\}$;
    end if
end for
$nums(s) = nums(s) + floor(rest/len(s))$;
$i = 0$;
while $rest > 0$ do
    $nums(s(i)) = nums(s(i)) + 1$;
    $i = i + 1$;
end while
```

The balance algorithm first calculates the average size of a specie μ_{avg}. It then generates an upper bound μ_{λ} for a specie size by multiplying it with λ. Then it resizes all species such that no specie is bigger than μ_{λ} , by removing individuals from the bigger species and uniformly adding them to species which were smaller than average (μ_{avg}) .

2.5.2 Mutate

Algorithm 9 Mutate (from [2])

```
1: Randomly generate two decimal numbers $type1$ and $type2$
   between 0 and 1;
2: if $type1 < per$ then
3:     if $type2 < 0.5$ then
4:         Generate the mutant $\boldsymbol{v}_i$ by DE/rand/1;
5:     else
6:         Generate the mutant $\boldsymbol{v}_i$ by DE/rand/2;
7:     end if
8: else
9:     if $type2 < 0.5$ then
10:        Generate the mutant $\boldsymbol{v}_i$ by DE/keypoint/1;
11:    else
12:        Generate the mutant $\boldsymbol{v}_i$ by DE/keypoint/2;
13:    end if
14: end if
```

Here, *per* is set as follows:

$$per = 1 - (evals/MaxFEs)^{\alpha} \tag{2.9}$$

where *evals* is the current number of evaluations consumed and *MaxFEs* is the maximum number of function evaluations allowed. α is a parameter that balances the ability of exploration and exploitation.

- DE/rand/1
$$v_i(g) = x_{r3}(g) + F * (x_{r1}(g) - x_{r2}(g)) \tag{2.10}$$
- DE/rand/2
$$v_i(g) = x_{r1}(g) + F * (x_{r2}(g) - x_{r3}(g)) + F * (x_{r4}(g) - x_{r5}(g)) \tag{2.11}$$
- DE/keypoint/1
$$v_i(g) = x_{r3}(g) + F * (x_{r1}(g) - x_{r2}(g)) \tag{2.12}$$
- DE/keypoint/2
$$v_i(g) = x_{r1}(g) + F * (x_{r2}(g) - x_{r3}(g)) + F * (x_{r4}(g) - x_{r5}(g)) \tag{2.13}$$

Here, $r1, r2, r3, r4, r5$ are indices random individuals from the sub-population. The parameter F (which is between 0 to 1) controls the effect of difference vectors. x_{kp} is an individual randomly selected from the key-points in the species.
To calculate key-points of a species, the NBC (Algorithm 2) is applied on that species.

Then the key-points are the resultant seeds, i.e. roots of the resultant subtrees.
When we execute NBC over a species, the parameter ϕ is denoted as ϕ_{kp}

2.6 A Niching Memetic Algorithm for Multi-Solution Traveling Salesman Problem ([6])

The multi-solution travelling salesman problem is based on permutations rather than continuous variables. We studied this paper to understand the formulation of this particular problem as well as to learn techniques that can be used to transform a continuous optimization algorithm to solve discrete permutation based problems. Techniques include a discretization based method and a random key based method.

2.7 Comparison between DSDE & FBK-DE

	DSDE	FBK-DE
Population size	80 to 300	250 to 1334
Clustering	Speciation clustering	NBC-Minsize
Cluster size	M = rand(4,20)	Decided by species balance strategy. Excess individuals are deleted. Additional individuals are generated.
Mutation strategy	DE/current-to-rand/1, DE/best/1	DE/rand/1, DE/rand/2, DE/keypoint/1, DE/keypoint/2
Mutation strategy selection	Based on inferior and superior division according to fitness	Random
Crossover	None in case of inferior Recombination in case of superior	Recombination
Selection strategy	Affinity Propagation Clustering followed by Adaptive Selection. OR Comparison with nearest parental individual	Better among child and parent
Archival strategy	Archive and reinitialise stagnated individuals	No archival

TABLE 2.1: Differences between DSDE and FBK-DE

Chapter 3

Design proposals

We'll be using FBK-DE as a base for our algorithm.

3.1 Nearest better clustering with temperature: softmax based cutting strategy

Assign a cutting probability to each edge which satisfies the cutting criteria (i.e. greater than $\mu *$ average length of all edges) using a softmax function with a certain temperature.

$$p(i) = \frac{e^{\frac{f(i)}{T}}}{\sum_j e^{\frac{f(j)}{T}}} \tag{3.1}$$

(Here T is the temperature and $f(i)$ is the length of the i-th edge)

Algorithm 10 Softmax Algorithm

```
1: N is the number of Edges
2: S = 0
3: P = empty array of size N
4: for i in 1..N do
5:     t = $e^{\frac{length(E_i)}{T}}$
6:     P[i] = t
7:     S += t
8: for i in 1..N do
9:     P[i] = P[i]/S
10: return P
```

Algorithm 11 Permute Softmax Algorithm

```
1: N is the number of Edges
2: P = Softmax(E) (**Algorithm 10**)
3: L = Generate indices without replacement using P as the array of sampling proba-
   bilities
4: ans = empty list
5: for i in 1..N do
6:     c = Edges[$L_i$]
7:     add c to ans
8: return ans
```

Now instead of sorting the edges in descending order of length (as in NBC-minsize: Algorithm 3), we sample from the edges (without replacement) using these probabilities.

Algorithm 12 NBC-Minsize-Temperature

```
1: Apply NBC-minsize (Algorithm 3) but instead of sorting on the 5th step, permute
   the edges using PermuteSoftmax (Algorithm 11)
```

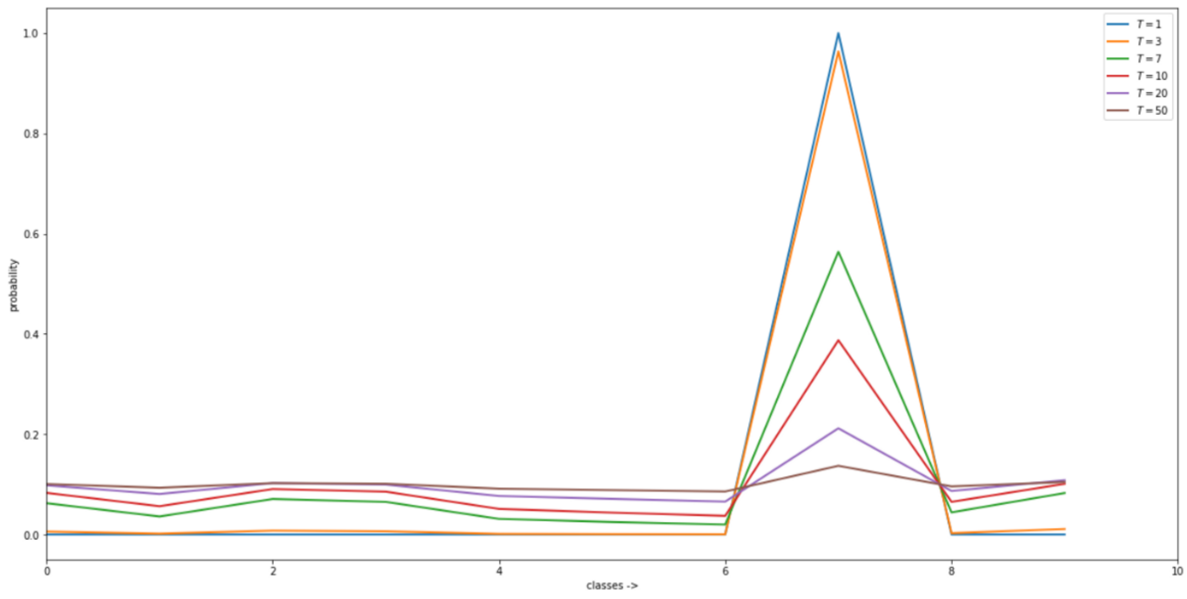

FIGURE 3.1: Softmax with different temperatures (from [7])

As $T \to \infty$, we get a "random" strategy as all edges have equal probability to be cut.
As $T \to 0$, we recover the original strategy of cutting the longest edge.
T can be dynamic as well (i.e. it can vary throughout the course of the algorithm)

3.2 Archival

We can use an archival strategy (inspired by **DSDE**) to detect and preserve stagnated/converged individuals in the archive. This saves additional function evaluations wasted in trying to mutate these individuals. The space freed by the archival of stagnated individuals can be filled with randomly initialised individuals, thereby increasing exploration.

3.3 Selection strategies

FBK-DE has a simple selection strategy that chooses one individual between the parent and the child. We can instead try to use selection strategies that work over the entire population at once by taking $2 * N$ individuals (old + new generation) as an input and selecting N individuals out of them.
Example of such selection strategies include adaptive selector and parent-child selector from the **DSDE** algorithm.

3.4 Stable mutation strategy

Algorithm 13 Stable Mutate algorithm

```
1: T = mutate(X) (Algorithm 9)
2: while T is not inside function domain do
3:     T = mutate(X) (Algorithm 9)
```

The mutated individual can sometimes go outside the function domain. We can reinitialise such individuals using our generation strategy or repeat the mutation process again until we get a suitable offspring.

3.5 Better new individual generation inspired by Multi-armed bandits

$$\begin{aligned} x_{rest} &= s_{seed} + N(0, 0.1) \\ x^j_{rest} &= max(x^j_{rest}, s^j_{lb}) \\ x^j_{rest} &= min(x^j_{rest}, s^j_{ub}) \end{aligned} \tag{3.2}$$

The generation strategy (which is used to generate new individuals in a single species when the size of the species is to be increased because of the balance strategy) used in **FBK-DE** is naive as it simply takes the best individual in a species (the 'seed') and adds some random noise to it. If the resulting individuals goes out of the species bounds, it is simply brought in by projecting it to the boundary surface. The points which are fixed in this way lie on the boundary, which adds a bias to the generation strategy.

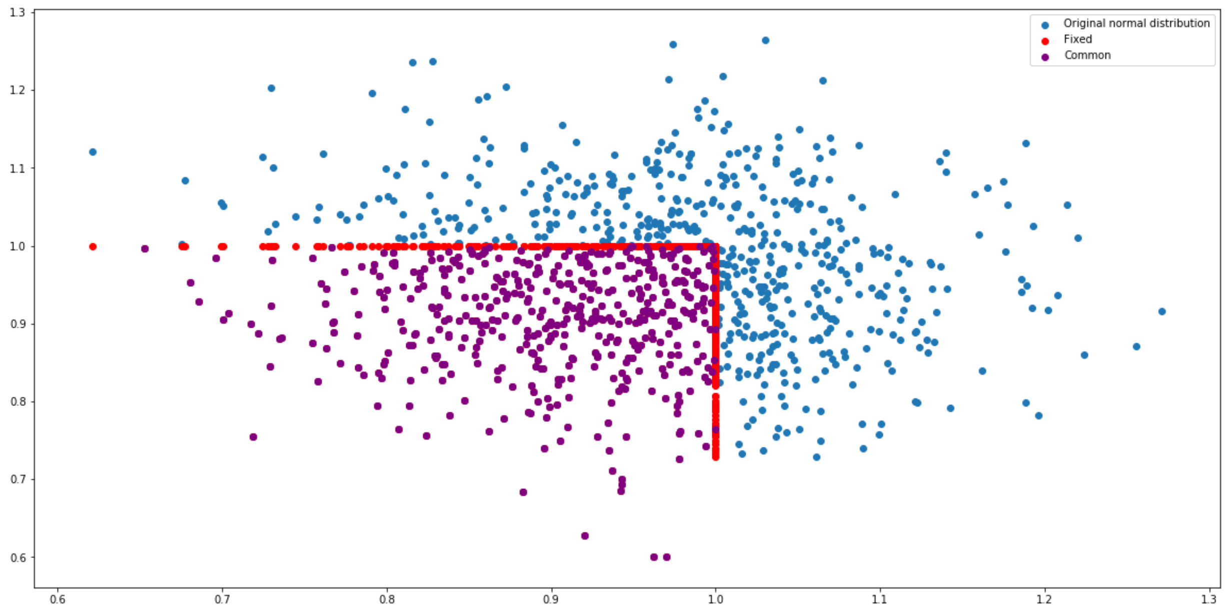

FIGURE 3.2: Illustration showing the fault in the original generate method

3.5.1 New Generate Method : NGM

$$\begin{aligned} x_{rest} &= s_{seed} + N(0, 0.1) \\ x^j_{rest} &= max(x^j_{rest}, Domain^j_{lb}) \\ x^j_{rest} &= min(x^j_{rest}, Domain^j_{ub}) \end{aligned} \tag{3.3}$$

Instead of bringing in the points based on the boundaries of the species, we can bring the point in based on the domain of the function.

Algorithm 14 New Generate Algorithm

```
1:  N = Number of individuals to be generated
2:  $s_{seed}$ = Best individual in the specie
3:  gen = []
4:  **for** i in 1..N **do**
5:      $x_{rest} = s_{seed} + N(0, 0.1)$
6:      **for** j in 1 to dimensions **do**
7:          $x^j_{rest} = max(x^j_{rest}, Domain^j_{LowerBound})$
8:          $x^j_{rest} = min(x^j_{rest}, Domain^j_{UpperBound})$
9:      append $x_{rest}$ to gen
10: **return** gen
```

3.5.2 Multi-armed Bandit : M

Algorithm 15 Goodness (UCB)

```
1: V = average fitness of all individuals in this sub-specie.
2: $N_i$ = number of individuals in this sub-specie
3: $N$ = number of Individuals in total
4: return $V + \sqrt{2 \log N / N_i}$
```

Algorithm 16 Best Sub-Specie Algorithm

```
1: Partition species using NBC (Algorithm 2)
2: Calculate goodness of each Sub-Specie (Algorithm 15)
3: return Best Sub-Specie
```

Algorithm 17 Multi Armed Bandit Generate Algorithm

```
1: N = Number of individuals to be generated
2: P = Select Best Sub-Specie using BSS (Algorithm 16)
3: gen = Generate new individuals using NGM(P, N) (Algorithm 14)
4: return gen
```

Instead of using a strategy that only uses the best individual, we can use a strategy that takes into account all individuals.
Multi-armed bandit problem are about allocating a fixed set of resources between alternative choices in a way that maximizes the expected reward. Each choice's properties are only partially known at the time of allocation, and become better understood by allocating resources to the choice [8][9]. This involves an exploration-exploitation trade-off dilemma.
We can use the best known strategies for solving multi-armed bandit problem for generating new individuals in a species by partitioning a species spatially using **NBC** and treating each partition as a different choice with different expected reward according to the fitness of the individuals present in that partition. We can then generate a new individual by sampling the partition chosen by a multi-armed bandit strategy, using **NGM**.

3.5.3 Multi Armed Bandit Iterative : MI

Algorithm 18 Multi Armed Bandit Iterative Generate Algorithm

```
1: N is the number of individuals to be generated
2: S is the Specie
3: for i in 1..N do
4:     gen = M(S,1) (Algorithm 17)
5:     Add gen to S
```

Instead of partitioning just once we can partition every time we generate a new individual, thus ensuring we are using the best estimates we have for all the clusters.
We can repeat this strategy the necessary number of times to add new individuals, this strategy is called **MI** (Multiarmed bandit Iterative) from here.

3.5.4 Multi Armed Bandit Repeated : MR

Algorithm 19 Multi Armed Bandit Repeated Generate Algorithm

```
1: N is the number of individuals to be generated
2: S is the Specie
3: while $S_{size} >$ Maximum Acceptable Size do
4:     T = BSS(S)
5:     if $S_{size} - T_{size} <$ Minimum Acceptable Rate then
6:         break
7:     S = T
8: gen = M(S,N) (Algorithm 17)
9: return gen
```

Instead of applying **BSS** just once we can repeatedly apply **BSS** to narrow down the individuals we should select in our final cluster, stopping when number of selected individuals fall below maximum acceptable size, or when the rate of decrease falls below a threshold minimum rate.

3.5.5 Multi Armed Bandit Iterative Repeated : MIR

Algorithm 20 Multi Armed Bandit Iterative Repeated Generate Algorithm

```
1: N is the number of individuals to be generated
2: S is the Specie
3: for i in 1..N do
4:     gen = MR(S,1) (Algorithm 19)
5:     Add gen to S
```

Combining both **MR** and **MI**, we can ensure we get the best clusters and we incorporate all the data we have while making decisions about where to generate new individuals

3.6 Final algorithm

After extensive experimentation (as described in Chapter 5), we determined the best parameter combinations for low-dim problems (1-15) and for high-dim problems (16-20). This is similar to how DSDE has 2 versions.

The two chosen versions (Final-1 and Final-2 respectively) are described below.

3.6.1 Flowchart

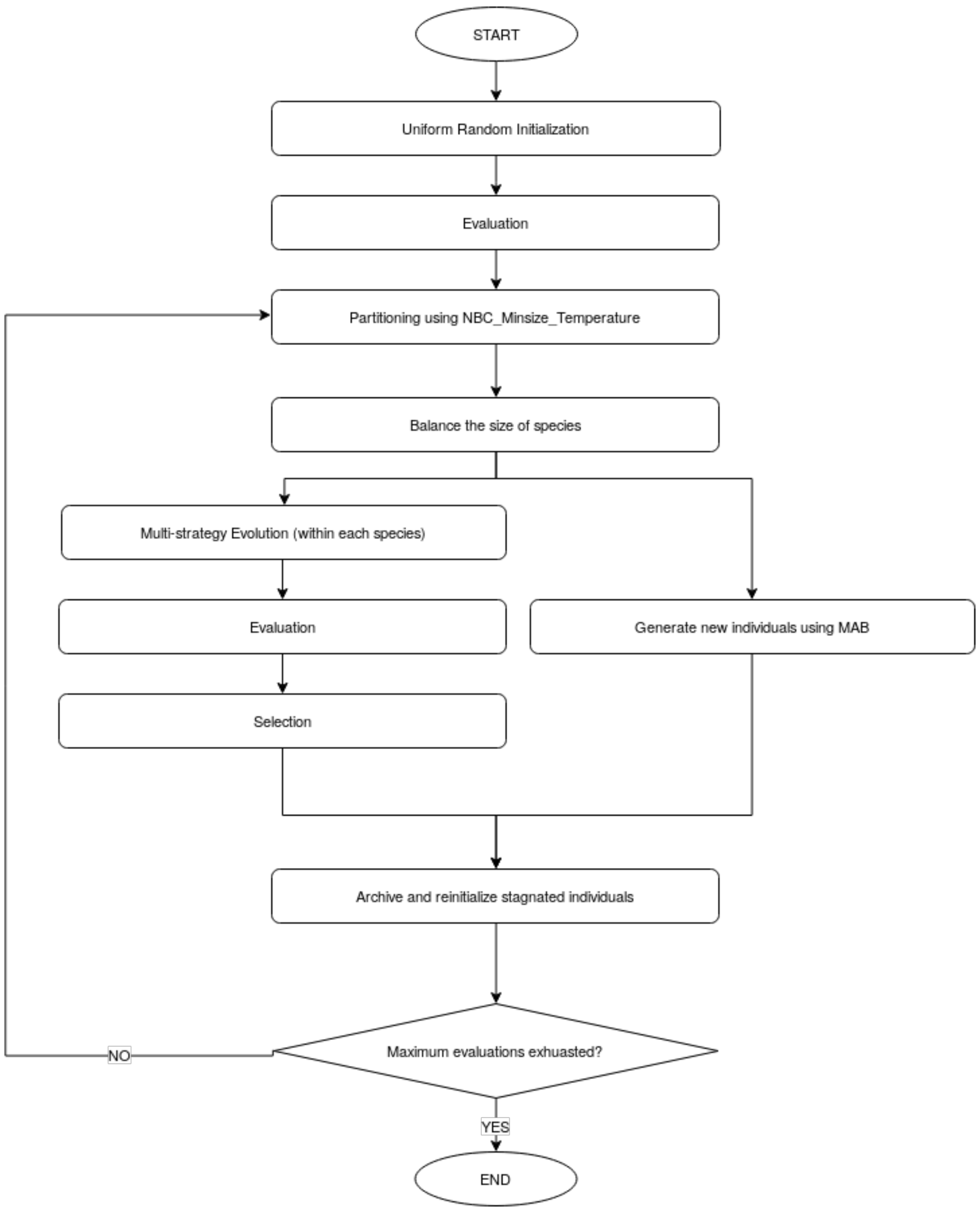
START
Uniform Random Initialization
Evaluation
Partitioning using NBC_Minsize_Temperature
Balance the size of species
Multi-strategy Evolution (within each species)
Evaluation
Generate new individuals using MAB
Selection
Archive and reinitialize stagnated individuals
Maximum evaluations exhuasted?
NO
YES
END

FIGURE 3.3: Flochart for the final algorithm

3.6.2 Psuedocode

The entire algorithm can be summarized as follows :

Algorithm 21 Final algorithm

```
1: Set MaxGens to 200 if dim is less than 5 else 300
2: Set the population size NP to MaxFes/MaxGens
3: Randomly generate the initial population P
4: g = 0
5: while termination condition is not satisfied do
6:     minsize = $min(5 + floor(\frac{g}{2}), max(10, 3 * dim))$
7:     Obtain the species by clustering using NBC Minsize Temp (Algorithm 12)
8:     Obtain the size of the species in array nums[] using balance(Algorithm 8)
9:     for each species si do
10:        for i=1 to min(len(s), nums(i)) do
11:            Generate $v_i(g)$ using stable mutate(Algorithm 13)
12:            Generate $u_i(g)$ using crossover equation(Algorithm 2.4)
13:            Put the better into $P(g + 1)$ using selection equation(Equation 2.5)
14:        if nums(i) > len(s) then
15:            Generate remaining individuals by using Generation Strategy(MI or MIR)(Algorithms 18 and 20) depending on the no. of dimension
16:            Insert the new individuals into $P(g + 1)$
17:     for each individual x do
18:        if x.StagnationCount >= T then
19:            Move x and those among its minsize closest neighbors whose fitness values are worse than x to the archive
20:            Reinitialize archived individuals
21:     $g = g + 1$
22: end
```

	ϕ (NBC in generate)	Generation strategy	Temperature (NBC in partitioning)	T (Archival)
Original	-	old	∞	∞
Final-1 (dim<5 : problems 1-15)	1	MI	0.5	30
Final-2 (dim>=5 : problems 16-20)	2	MIR	0.5	60

TABLE 3.1: Parameters for final-1,final-2 and original

Parameters	Value
λ	2
ϕ_{kp}	2
α	0.5
CR	0.9
F	0.2 0.8 (one differential vector) 0.5 (two differential vectors)
Max Acceptable Size	5
Min Acceptable Rate	5

TABLE 3.2: Common parameters

3.6.3 Complexity analysis

Our proposed algorithm contains four significant mechanisms: NBC-minsize-temp, species balance strategy, keypoint-based mutation operators used in stable mutate algorithm and the multi armed bandits inspired generate strategy. Here, we analyze the time complexity of each mechanism.

In NBC-minsize-temp (Algorithm 12), there are three time-consuming operations. First is to construct a spanning tree which involves calculating the distance between each pair of points. The time complexity of this operation is $O(n^2)$. The second is to check whether an edge should be cut off according to minsize criteria. The time complexity of this operation is $O(n)$. The third is to calculate softmax (Algorithm 11) over the array of edges. The number of edges in the spanning tree is $n-1$, so calculating softmax over the array takes $O(n)$ time and then sampling according to the probabilities (without replacement) takes $O(n^2)$ time. Thus, the complexity of NBC-minsize-temp is $O(n^2)$.

From Algorithm 8, we can see that the time complexity of the species balance strategy is $O(n)$, as the number of species is certainly less than n.

In the keypoint-based mutation operators (Algorithm 9 as referenced in algorithm 13), we need execute the original NBC (Algorithm 2) to identify the keypoints. However, there is no need to calculate the distances again, as we already have the pairwise distances obtained in NBC-minsize-temp. Because of this, the time complexity of finding the keypoints is $O(n)$, and DE operators over each species requires at most $O(n)$.

Now consider the generation strategy. To generate m individuals, the new generate method (NGM : Algorithm 14) takes $O(m)$ time.

In what follows, note that s is the original species size and m is the number of new individuals to be generated.

The Multi Armed Bandit Generate Algorithm (17) uses the original NBC over a species of size s which takes time $O(s)$ as the distances are already known. Further, the average fitness of all sub-species is calculated which also takes time $O(s)$ as the sum of sizes of all subspecies is s. Finally, to generate m individuals, NGM is called which takes time $O(m)$. Thus the total time is $O(m+s)$.

Now for the Multi Armed Bandit Iterative Generate Algorithm (18), the Algorithm 17 is called m times to generate 1 individual each time. This requires extra distance computations of order $O(s + (s+1) + ..(s+m))$ which is $O(m * s + m^2)$. In total, the complexity is $(O(1+s) + ... + O(1 + (m+s))) + O(m * s + m^2)$ or equivalently $O(m * s + m^2)$.

In the Multi Armed Bandit Repeated Generate Algorithm (19), the distances are already known and NBC is repeatedly applied giving time complexity $O(s + (s - MAR) + (s - 2 * MAR) + ... + (MAS))$ which is less than $O(s^2/MAR)$ (with $MAR > 1$). Finally, to generate m individuals, NGM is called which takes time $O(m)$. Thus the total time is $O(m + s^2)$.

For the Multi Armed Bandit Iterative Repeated Algorithm (20), the Algorithm 19 is called m times to generate 1 individual each time. This requires extra distance computations of order $O(s + (s+1) + ..(s+m))$ which is $O(m * s + m^2)$. In total, the complexity is $(O(1 + (s)^2) + ... + O(1 + (m+s)^2)) + O(m * s + m^2)$ or equivalently $O(m^3 + m * s^2)$.

In our proposed algorithm for low dimensional problems (Final$_1$), we use Multi Armed Bandit Iterative Generate Algorithm (18). The complexity of this was shown to be $O(m * s + m^2)$. For the worst case, m and s can be both of $O(n)$, giving a worst case complexity of $O(n^2)$. (Note however that this is the worst case, on average we expect $m << n$ and s to be $O(n)$, giving an average case complexity closer to $O(n)$.)

In our proposed algorithm for high dimensional problems (Final$_2$), we use Multi Armed Bandit Iterative Repeated Generate Algorithm (20). The complexity of this was shown to be $O(m^3 + m * s^2)$. For the worst case, m and s can be both of $O(n)$, giving a worst case complexity of $O(n^3)$. (Note however that this is the worst case, on average we expect $m << n$ and s to be $O(n)$, giving an average case complexity closer to $O(n^2)$.)

Therefore, the worst case time complexity of Final$_1$ is $O(MaxGen * n^2)$ and that of Final$_2$ is $O(MaxGen * n^3)$, where $MaxGen$ is the maximum evolutionary generations allowed and n is the population size.

Chapter 4

Implementation details

We implemented all the algorithms in Python 3. This includes the 3 current state of the art algorithms: DSDE (Algorithm 4), DSDE-C (Algorithm 5), FBK-DE (Algorithm 7) and our own proposed algorithm (Algorithm 21). We also used scientific computation libraries such as numpy, scipy and scikit-learn.

Some parts were common in all three algorithms, so we separated them into a separate module to facilitate reuse and reduce duplication of code. The common parts include:

1. Agent class to represent an individual
2. Uniform random initialization within the given bounds
3. Code used to interface with the CEC-2013 benchmarking suite
4. Crossover operator with a parameter
5. Termination condition exception
6. Code to save and load populations
7. Code for judging the final population.

We added auto-evaluation to the Agent class so that each individual's fitness is calculated as soon as it is initialized. We made individuals immutable and added a stagnation count property to the Agent class to keep track of the number of generations that the individual has survived (this is used in archiving).

We parallelized the code using numpy matrices and arrays whenever possible. Data structures like min-heaps were used to make the algorithms efficient. Scipy (a scientific computation library) was used for fast calculation of softmax over arrays (used for deciding the order in which to cut edges in NBC with temperature: Algorithm 12).

We implemented a tester function to test the algorithms on multiple functions, with different parameters. The tester function writes peak ratio and success ratio scores for all accuracy levels on the specified problems to a file. A rolling average is used so that the testing can be done for any number of runs.

Chapter 5

Experimentation

We performed extensive testing to determine the best combination of strategies and parameters for our algorithm.

As part of our work we also created a framework to parallelize the testing on such a big dataset. Testing was parallelized using python's multiprocessing module. A 32 core server was used to run 32 instances of the algorithm at once. This provides a linear speedup which helped us reduce our testing time by over 32 times. We added command line arguments to our algorithm to test various parameter combinations and facilitate parallel testing.

During initial experimentation, each combination of parameters was run 5 times and the result was recorded in a separate file (whose name is simply the concatenation of all parameter values).

The following details were recorded for each parameter combination: (These metrics were described in Chapter 2, section 2.3.2)

1. Average peak ratios (see Equation 2.6) and associated variance for five different accuracy levels
2. Average success ratios (see Equation 2.7) and associated variance for five different accuracy levels

The following parameters were tested:

1. ϕ_{gen} (NBC in generate)
2. Generation strategy (old,new,m,mi,mir)
3. T (Archival)
4. Temperature (for softmax, used in NBC-minsize-temp)
5. α
6. Generation multiplier
7. M_{factor} (Archival): Decides the number of neighbours to consider when archiving a stagnated individual. M = (1-M_{factor}) * minsize + M_{factor}* average size of species.
8. Maximum acceptable size (used in MIR generation strategy)
9. Minimum acceptable rate (used in MIR generation strategy)

The results were analysed using various criteria:

1. **Average PR**: The average peak ratio obtained by a parameter combination across all problems
2. **Average relative PR** (=actual-PR/fbk-de-PR): The average relative peak ratio (=actual-PR/fbk-de-PR) obtained by a parameter combination across all problems. This criteria considers relative improvement rather than absolute improvement.
3. **Average rank**: The average of all ranks obtained by a parameter combination on all problems. The lesser the better. For example, if a parameter combination ranks first on half the problems (i.e. it's the best performing combination for those problems) and ranks second on the remaining half of the problems, then the average rank for that parameter combination would be 1.5
4. **Points** (=number of problems where PR was better than that of fbk-de): This criteria simply counts the number of times a particular parameter combination performed better than normal FBK-DE.

Analysis was done by writing Python scripts. After this process, the best parameter combinations were identified for low-dim problems (dim<5 : problems 1-15) and for high-dim problems (dim>=5 : problems 16-20). This is similar to how DSDE has 2 versions.

The two chosen versions (Final-1 and Final-2 respectively) plus the replicated version of FBK-DE were run over 30 times and the average peak ratios & associated variance for five different accuracy levels and also the average success ratios & associated variance for five different accuracy levels were recorded.

Chapter 6

Results

6.1 Performance of FBK-DE (replicated)

Here we present the results of our replication of the previous state of the art FBK-DE, the values shown below are the averages after running each combination 30 times, to reduce variance caused by randomness

6.1.1 Peak Ratios at various accuracy levels

	$1e^{-1}$	$1e^{-2}$	$1e^{-3}$	$1e^{-4}$	$1e^{-5}$
F1	1.000	1.000	1.000	1.000	1.000
F2	1.000	1.000	1.000	1.000	1.000
F3	1.000	1.000	1.000	1.000	1.000
F4	1.000	1.000	1.000	1.000	1.000
F5	1.000	1.000	1.000	1.000	1.000
F6	0.907	0.904	0.898	0.896	0.896
F7	0.753	0.744	0.744	0.744	0.744
F8	0.590	0.581	0.573	0.569	0.563
F9	0.373	0.365	0.363	0.361	0.359
F10	1.000	1.000	1.000	1.000	1.000
F11	1.000	1.000	1.000	1.000	1.000
F12	0.883	0.883	0.883	0.883	0.883
F13	1.000	1.000	1.000	1.000	1.000
F14	0.856	0.850	0.850	0.850	0.850
F15	0.658	0.654	0.654	0.650	0.650
F16	0.667	0.667	0.667	0.667	0.667
F17	0.569	0.569	0.569	0.569	0.569
F18	0.667	0.667	0.667	0.667	0.667
F19	0.500	0.472	0.472	0.472	0.472
F20	0.417	0.417	0.417	0.417	0.403

TABLE 6.1: FBK-DE (replicated): Peak Ratios at various accuracy levels

6.1.2 Success Ratios at various accuracy levels

	$1e^{-1}$	$1e^{-2}$	$1e^{-3}$	$1e^{-4}$	$1e^{-5}$
F1	1.000	1.000	1.000	1.000	1.000
F2	1.000	1.000	1.000	1.000	1.000
F3	1.000	1.000	1.000	1.000	1.000
F4	1.000	1.000	1.000	1.000	1.000
F5	1.000	1.000	1.000	1.000	1.000
F6	0.100	0.100	0.067	0.067	0.067
F7	0.000	0.000	0.000	0.000	0.000
F8	0.000	0.000	0.000	0.000	0.000
F9	0.000	0.000	0.000	0.000	0.000
F10	1.000	1.000	1.000	1.000	1.000
F11	1.000	1.000	1.000	1.000	1.000
F12	0.167	0.167	0.167	0.167	0.167
F13	1.000	1.000	1.000	1.000	1.000
F14	0.167	0.133	0.133	0.133	0.133
F15	0.000	0.000	0.000	0.000	0.000
F16	0.000	0.000	0.000	0.000	0.000
F17	0.000	0.000	0.000	0.000	0.000
F18	0.000	0.000	0.000	0.000	0.000
F19	0.000	0.000	0.000	0.000	0.000
F20	0.000	0.000	0.000	0.000	0.000

TABLE 6.2: FBK-DE (replicated): Success Ratios at various accuracy levels

6.2 Performance of Final-1 algo

Here, we present the results of Final-1 on the benchmark dataset, This algorithm performs better than the previous state of the on low dimensional problems (dim<5 : problems 1-15). The values shown below are the averages after running each combination 30 times, to reduce variance caused by randomness.

6.2.1 Peak Ratios at various accuracy levels

	$1e^{-1}$	$1e^{-2}$	$1e^{-3}$	$1e^{-4}$	$1e^{-5}$
F1	1.000	1.000	1.000	1.000	1.000
F2	1.000	1.000	1.000	1.000	1.000
F3	1.000	1.000	1.000	1.000	1.000
F4	1.000	1.000	1.000	1.000	1.000
F5	1.000	1.000	1.000	1.000	1.000
F6	0.998	0.998	0.998	0.996	0.989
F7	0.998	0.838	0.830	0.826	0.822
F8	0.712	0.707	0.703	0.699	0.697
F9	0.856	0.434	0.420	0.406	0.400
F10	1.000	1.000	1.000	1.000	1.000
F11	1.000	1.000	1.000	1.000	1.000
F12	0.929	0.929	0.925	0.925	0.925
F13	1.000	1.000	1.000	1.000	1.000
F14	0.894	0.861	0.861	0.861	0.861
F15	0.829	0.717	0.717	0.717	0.717
F16	0.889	0.667	0.667	0.667	0.667
F17	0.625	0.625	0.625	0.625	0.625
F18	0.889	0.667	0.611	0.5	0.5
F19	0.583	0.417	0.417	0.417	0.417
F20	0.542	0.5	0.417	0.417	0.375

TABLE 6.3: Final-1 algo: Peak Ratios at various accuracy levels

6.2.2 Success Ratios at various accuracy levels

	$1e^{-1}$	$1e^{-2}$	$1e^{-3}$	$1e^{-4}$	$1e^{-5}$
F1	1.000	1.000	1.000	1.000	1.000
F2	1.000	1.000	1.000	1.000	1.000
F3	1.000	1.000	1.000	1.000	1.000
F4	1.000	1.000	1.000	1.000	1.000
F5	1.000	1.000	1.000	1.000	1.000
F6	0.967	0.967	0.967	0.933	0.800
F7	0.967	0.000	0.000	0.000	0.000
F8	0.000	0.000	0.000	0.000	0.000
F9	0.067	0.000	0.000	0.000	0.000
F10	1.000	1.000	1.000	1.000	1.000
F11	1.000	1.000	1.000	1.000	1.000
F12	0.433	0.433	0.400	0.400	0.400
F13	1.000	1.000	1.000	1.000	1.000
F14	0.367	0.167	0.167	0.167	0.167
F15	0.367	0.000	0.000	0.000	0.000
F16	0.000	0.000	0.000	0.000	0.000
F17	0.000	0.000	0.000	0.000	0.000
F18	0.000	0.000	0.000	0.000	0.000
F19	0.000	0.000	0.000	0.000	0.000
F20	0.000	0.000	0.000	0.000	0.000

TABLE 6.4: Final-1 algo: Success Ratios at various accuracy levels

6.2.3 Comparison between Final-1 and FBK-DE

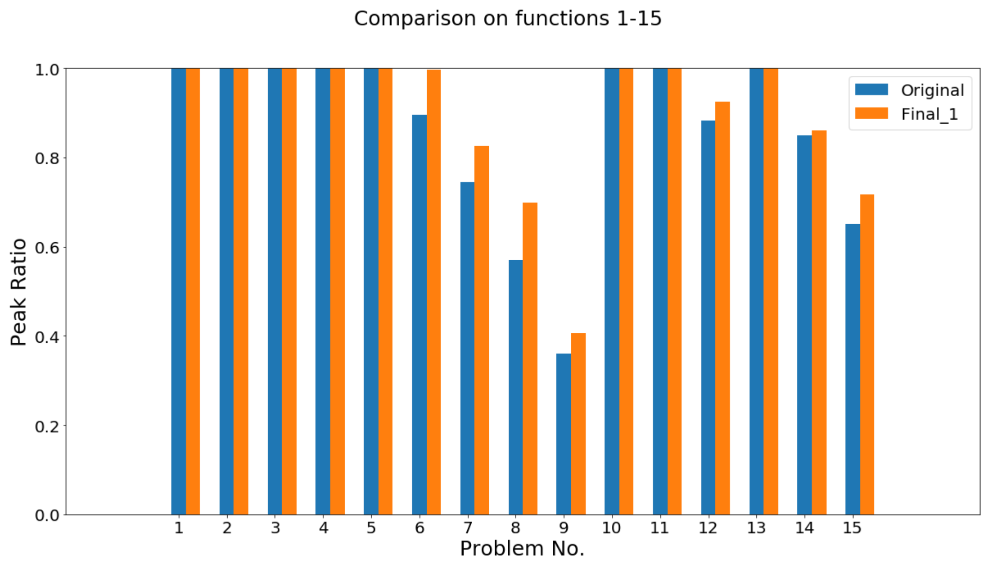

FIGURE 6.1: Comparison between Final-1 and FBK-DE

From the comparison above we can see that Final-1 improves upon FBK-DE on the first 16 problems, whenever FBK-DE didn't already perfectly solve the problem.

6.3 Performance of Final-2 algo

Here we present the results of Final-2 on the benchmark dataset. This algorithm performs better than the previous state of the art on high dimensional problems (dim>=5 : problems 16-20). the values shown below are the averages after running each combination 30 times, to reduce variance caused by randomness.

6.3.1 PR at various accuracy levels

	$1e^{-1}$	$1e^{-2}$	$1e^{-3}$	$1e^{-4}$	$1e^{-5}$
F1	1.000	1.000	1.000	1.000	1.000
F2	1.000	1.000	1.000	1.000	1.000
F3	1.000	1.000	1.000	1.000	1.000
F4	1.000	1.000	1.000	1.000	1.000
F5	1.000	1.000	1.000	1.000	1.000
F6	0.977	0.977	0.958	0.954	0.944
F7	0.833	0.833	0.833	0.833	0.833
F8	0.691	0.684	0.682	0.677	0.672
F9	0.494	0.443	0.414	0.390	0.379
F10	1.000	1.000	1.000	1.000	1.000
F11	1.000	1.000	1.000	1.000	1.000
F12	0.906	0.906	0.906	0.906	0.906
F13	1.000	1.000	1.000	1.000	1.000
F14	0.931	0.903	0.903	0.903	0.903
F15	0.823	0.677	0.677	0.677	0.677
F16	0.907	0.685	0.685	0.685	0.685
F17	0.708	0.653	0.653	0.653	0.653
F18	0.796	0.667	0.667	0.667	0.667
F19	0.653	0.514	0.514	0.514	0.514
F20	0.458	0.444	0.444	0.444	0.444

TABLE 6.5: Final-2 algo: Peak Ratios at various accuracy levels

6.3.2 SR at various accuracy levels

	$1e^{-1}$	$1e^{-2}$	$1e^{-3}$	$1e^{-4}$	$1e^{-5}$
F1	1.000	1.000	1.000	1.000	1.000
F2	1.000	1.000	1.000	1.000	1.000
F3	1.000	1.000	1.000	1.000	1.000
F4	1.000	1.000	1.000	1.000	1.000
F5	1.000	1.000	1.000	1.000	1.000
F6	0.750	0.750	0.417	0.333	0.250
F7	0.000	0.000	0.000	0.000	0.000
F8	0.000	0.000	0.000	0.000	0.000
F9	0.000	0.000	0.000	0.000	0.000
F10	1.000	1.000	1.000	1.000	1.000
F11	1.000	1.000	1.000	1.000	1.000
F12	0.250	0.250	0.250	0.250	0.250
F13	1.000	1.000	1.000	1.000	1.000
F14	0.667	0.500	0.500	0.500	0.500
F15	0.417	0.000	0.000	0.000	0.000
F16	0.667	0.000	0.000	0.000	0.000
F17	0.111	0.000	0.000	0.000	0.000
F18	0.222	0.000	0.000	0.000	0.000
F19	0.111	0.000	0.000	0.000	0.000
F20	0.000	0.000	0.000	0.000	0.000

TABLE 6.6: Final-2 algo: Success Ratios at various accuracy levels

6.3.3 Comparison between Final-2 and FBK-DE

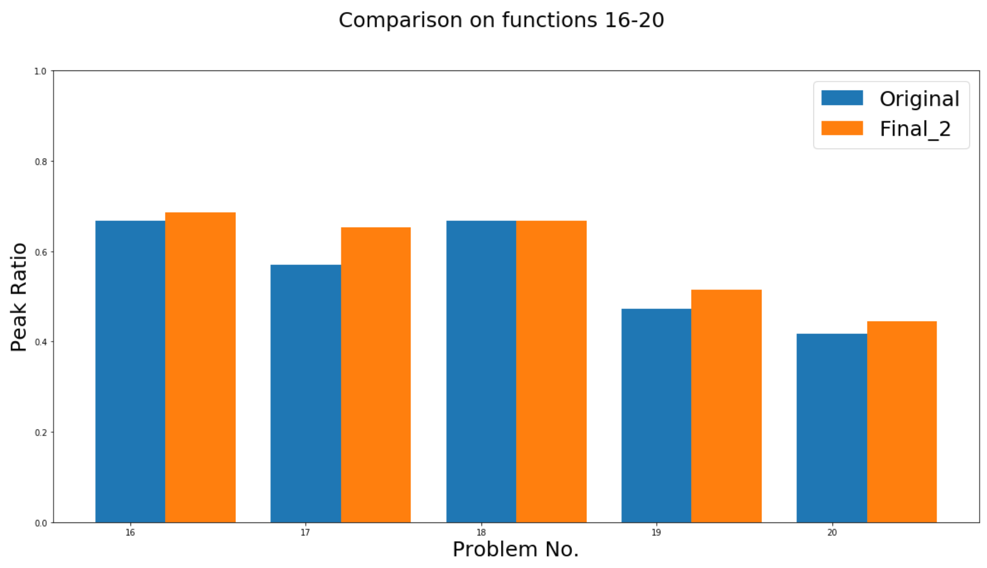

FIGURE 6.2: Comparison between Final-2 algo and FBK-DE

From the comparison above we can see that Final-2 improves upon FBK-DE on high dimensional problems.

6.4 Complete comparison at accuracy 10^{-4}

Below we present the side by side comparison of Final-1 , Final-2 and FBK-DE (Original), at the accuracy level of 10^{-4}. As we can see, Final-1 performs the best on low dim problems (dim<5 : problems 1-15) while Final-2 performs the best on high dim problems (dim>=5 : problems 16-20) .

	Final-1		Original		Final-2	
	PR	SR	PR	SR	PR	SR
F1	1.000	1.000	1.000	1.000	1.000	1.000
F2	1.000	1.000	1.000	1.000	1.000	1.000
F3	1.000	1.000	1.000	1.000	1.000	1.000
F4	1.000	1.000	1.000	1.000	1.000	1.000
F5	1.000	1.000	1.000	1.000	1.000	1.000
F6	**0.996**	**0.933**	0.896	0.067	0.954	0.333
F7	0.826	0.000	0.744	0.000	**0.833**	**0.000**
F8	**0.699**	**0.000**	0.569	0.000	0.677	0.000
F9	**0.406**	**0.000**	0.361	0.000	0.390	0.000
F10	1.000	1.000	1.000	1.000	1.000	1.000
F11	1.000	1.000	1.000	1.000	1.000	1.000
F12	**0.925**	**0.400**	0.883	0.167	0.906	0.250
F13	1.000	1.000	1.000	1.000	1.000	1.000
F14	0.861	0.167	0.850	0.133	**0.903**	**0.500**
F15	**0.717**	**0.000**	0.650	0.000	0.677	0.000
F16	0.667	0.000	0.667	0.000	**0.685**	**0.000**
F17	0.625	0.000	0.569	0.000	**0.653**	**0.000**
F18	0.500	0.000	0.667	0.000	**0.667**	**0.000**
F19	0.417	0.000	0.472	0.000	**0.514**	**0.000**
F20	0.417	0.000	0.417	0.000	**0.444**	**0.000**

TABLE 6.7: Comparison between Final-1, FBK-DE and Final-2

6.4.1 Comparison between Final-1, Final-2 and FBK-DE

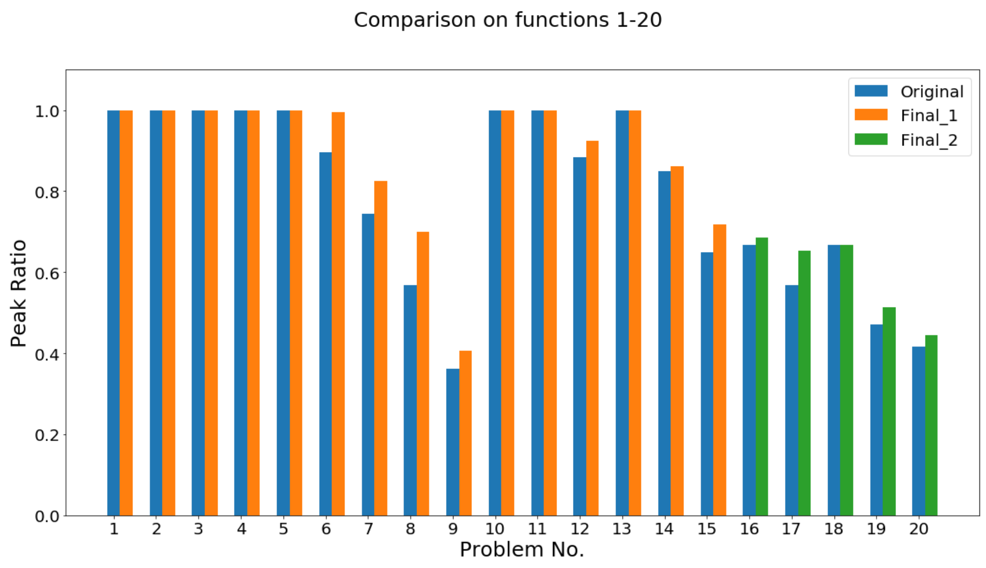

FIGURE 6.3: Comparison between Final-1, FBK-DE and Final-2

Excluding the problems where FBK-DE performed perfectly, we obtain a 22% reduction in error on average and 9% improvement in Peak Ratio on average.

Chapter 7

Conclusion

We've studied evolutionary algorithm concepts. We've studied and implemented 3 algorithms: **DSDE**, **DSDE-C** *(Dual-Strategy Differential Evolution With Affinity Propagation Clustering)* & **FBK-DE** *(formulation, balance, and keypoint - a Differential Evolution algorithm using Nearest-Better Clustering)*.

We combined the concepts learned from the three algorithms that we've implemented and propose novel strategies for clustering, generation & archival to create a new algorithm.

Exhaustive testing was done for hyper-parameter optimization as well as for choosing the best strategies.

We present a multi armed bandit inspired algorithm which furthers the state of the art in solving multi-modal optimisation problems. We are releasing the source code of our algorithm and also our reproduction of the previous state of the art.

We present two different versions of our algorithm which show different characteristics on different kinds of problem types. In particular, the first version is suitable for low dimensional problems while the latter one is meant for higher dimensional problems.

We are able to improve upon the replicated FBK-DE results. Excluding the problems where FBK-DE performed perfectly (in which our algorithm also performs perfectly), we obtain a 22% reduction in error on average and 9% improvement in Peak Ratio on average.

We are also releasing accessible resources (this thesis as well as a presentation explaining our project) that would help future researchers to learn about the problem and help contribute to the field. Also, a research paper explaining our proposed algorithm is currently being worked on.

The current formulation of the algorithm requires careful hyperparameter tuning, finding formulations that are hyperparameter insensitive is the future direction of research. Another way to extend our work would be to try other kinds of multi armed bandit strategies to further improve the performance.

Appendix A

The CEC-2013 benchmark suite

A.1 Description

In this section, we briefly describe the dataset used to test multimodal optimization algorithms. We've used the CEC-2013 benchmark suite [3] for testing our algorithm as it is the standard benchmark used in the field of evolutionary multimodal optimization. It consists of 20 multimodal functions of different dimensions. Certain characteristics of functions available in dataset are described in Table A.1.

The first 10 benchmark functions are simple, well known and widely used functions, largely based on earlier studies. The remaining benchmark functions are more complex and follow the paradigm of composition functions. All of them are maximization problems and have a given limit of maximum number of function evaluations (MaxFEs). The benchmark also includes a method to check how many optima we found within a given accuracy level. The peak height in the dataset is the height of the optimal solutions to be found, while niche radius is the minimum distance between the two optimal solutions such that two optimal solutions found can be differentiated.

Index	Dimension	Available Peaks	Peak Height	Niche Radius	Max FEs
1	1D	2	200.0	0.01	5.0E+4
2	1D	5	1.0	0.01	5.0E+4
3	1D	1	1.0	0.01	5.0E+4
4	2D	4	200.0	0.01	5.0E+4
5	2D	2	1.03163	0.5	5.0E+4
6	2D	18	186.731	0.5	2.0E+4
7	2D	36	1.0	0.2	2.0E+4
8	3D	81	-2.0	0.5	4.0E+4
9	3D	216	0	0.2	4.0E+4
10	2D	12	0	0.01	2.0E+4
11	2D	6	0	0.01	2.0E+4
12	2D	8	0	0.01	2.0E+4
13	2D	6	0	0.01	2.0E+4
14	3D	6	0	0.01	4.0E+4
15	3D	8	0	0.01	4.0E+4
16	5D	6	0	0.01	4.0E+4
17	5D	8	0	0.01	4.0E+4
18	10D	6	0	0.01	4.0E+4
19	10D	8	0	0.01	4.0E+4
20	20D	8	0	0.01	4.0E+4

TABLE A.1: Function Characteristics

A.2 Examples of functions (from [3])

In all these functions, the dimensionality refers to the number of dimensions in the input. The output (called the 'fitness') is always a 1 dimensional real number.

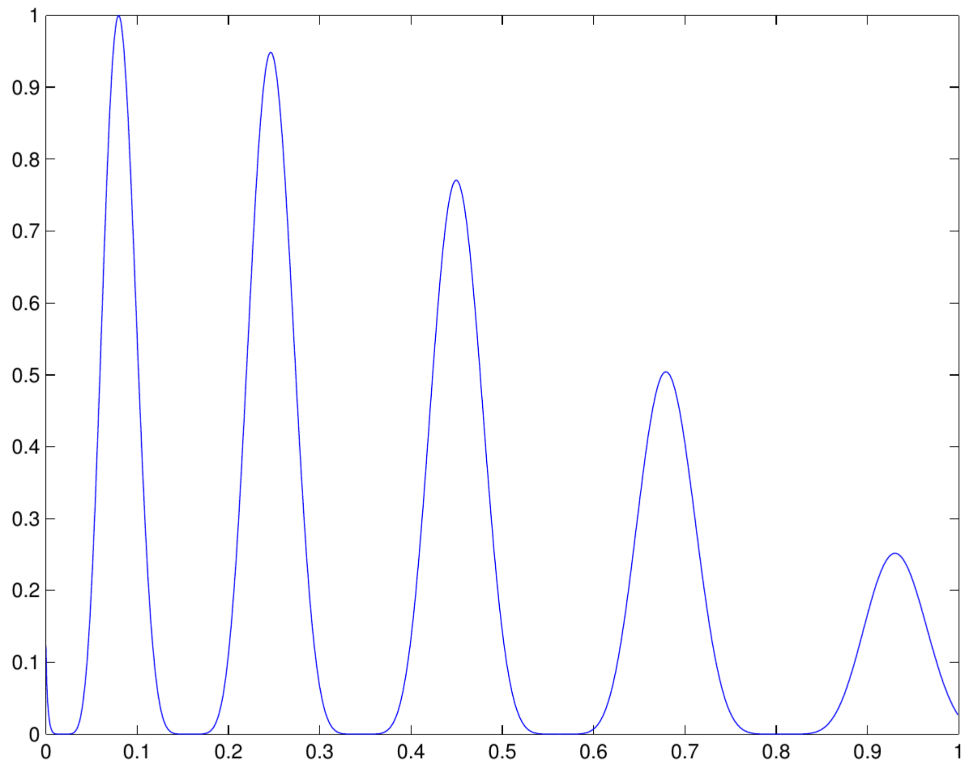

FIGURE A.1: F3 : Uneven Decreasing Maxima (1D)

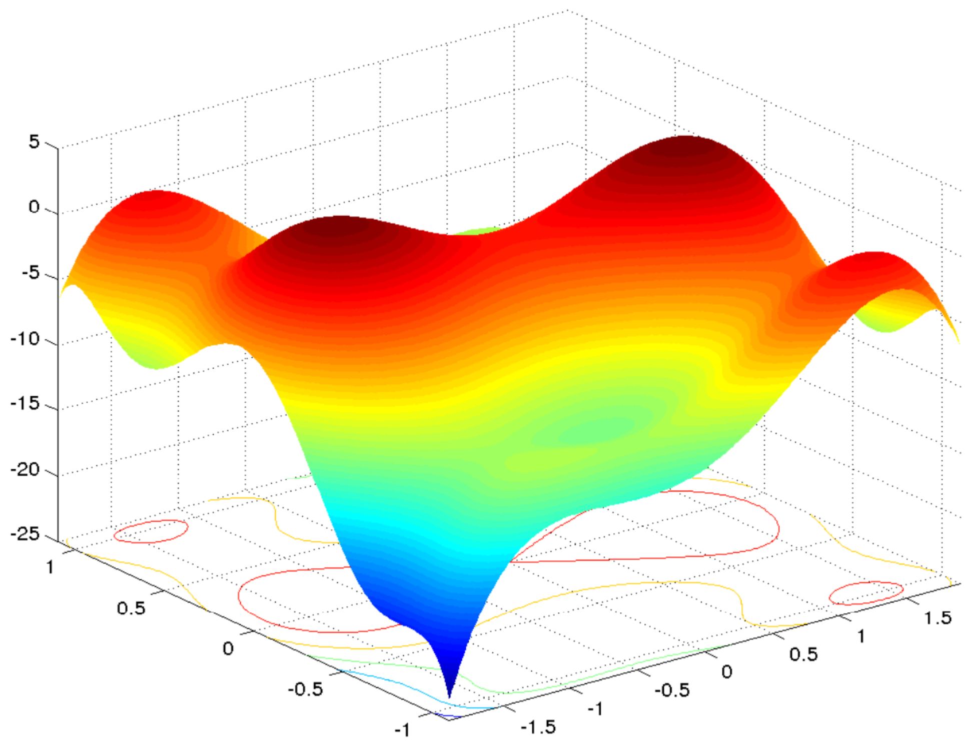

FIGURE A.2: F5 : Six-Hump Camel Back (2D)

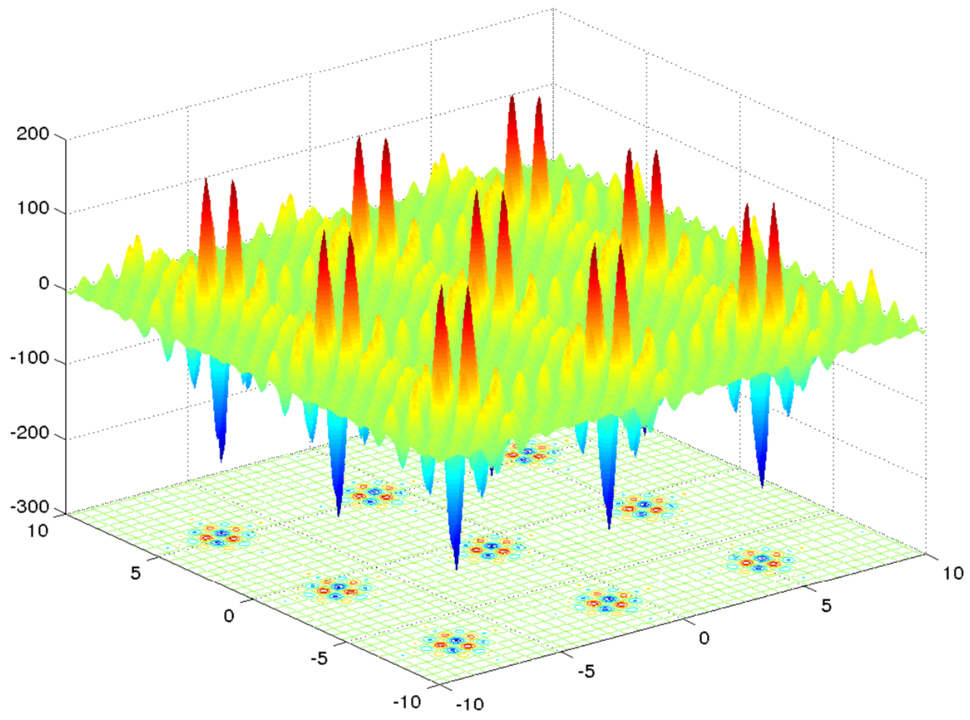

FIGURE A.3: F6 : Shubert (2D)

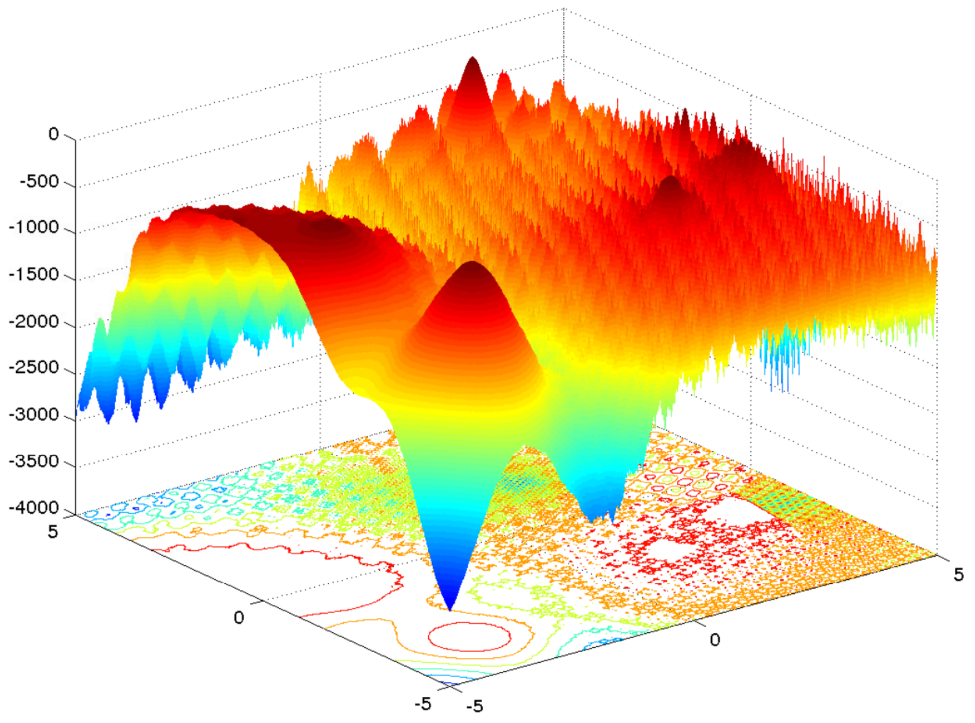

FIGURE A.4: F12 : Composition Function 4 (2D version)

Bibliography

[1] Zi-Jia Wang et al. "Dual-Strategy Differential Evolution With Affinity Propagation Clustering for Multimodal Optimization Problems". In: *IEEE Transactions on Evolutionary Computation* 22.6 (2018), 894–908. DOI: 10.1109/tevc.2017.2769108.

[2] Xin Lin, Wenjian Luo, and Peilan Xu. "Differential Evolution for Multimodal Optimization With Species by Nearest-Better Clustering". In: *IEEE Transactions on Cybernetics* (2019), 1–14. DOI: 10.1109/tcyb.2019.2907657.

[3] Xiaodong Li, Andries Petrus Engelbrecht, and Michael G. Epitropakis. "Benchmark Functions for CEC'2013 Special Session and Competition on Niching Methods for Multimodal Function Optimization'". In: 2013.

[4] MIKE PREUSS. *MULTIMODAL OPTIMIZATION BY MEANS OF EVOLUTIONARY ALGORITHMS*. SPRINGER, 2016.

[5] Mikeagn. *mikeagn/CEC2013*. URL: https://github.com/mikeagn/CEC2013.

[6] Ting Huang et al. "A Niching Memetic Algorithm for Multi-Solution Traveling Salesman Problem". In: *IEEE Transactions on Evolutionary Computation* (2019), 1–1. DOI: 10.1109/tevc.2019.2936440.

[7] Ravindra Kompella. *Tap into the dark knowledge using neural nets - Knowledge distillation*. 2018. URL: https://towardsdatascience.com/knowledge-distillation-and-the-concept-of-dark-knowledge-8b7aed8014ac.

[8] J. C. Gittins. *Multi-armed bandit allocation indices*. John Wiley and Sons, 1989.

[9] D. A. Berry and B. Fristedt. *Bandit problems ; sequential allocation of experiments*. Chapman and Hall, 1985.