LARGE SCALE ESTIMATION OF DISTRIBUTION ALGORITHMS FOR CONTINUOUS OPTIMISATION

by

MOMODOU LAMIN SANYANG

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
October 2017

Dedication

To my Mum, Dad, wife and kids

Abstract

Modern real world optimisation problems are increasingly becoming large scale. However, searching in high dimensional search spaces is notoriously difficult. Many methods break down as dimensionality increases and Estimation of Distribution Algorithm (EDA) is especially prone to the curse of dimensionality. In this thesis, we device new EDA variants that are capable of searching in large dimensional continuous domains. We in particular (i) investigated heavy tails search distributions, (ii) we clarify a controversy in the literature about the capabilities of Gaussian versus Cauchy search distributions, (iii) we constructed a new way of projecting a large dimensional search space to low dimensional subspaces in a way that gives us control of the size of covariance of the search distribution and we develop adaptation techniques to exploit this and (iv) we proposed a random embedding technique in EDA that takes advantage of low intrinsic dimensional structure of problems. All these developments avail us with new techniques to tackle high dimensional optimisation problems.

Acknowledgements

I would like to start by expressing my immense gratitude to almighty God for giving me the strength, health, and determination to successfully complete this thesis.

First and foremost, I want to thank my lovely wife, Anna Mbye Sanyang for her untiring and constant source of support, love, patience and good humour which kept me going throughout the during of this PhD journey. Thanks Anna mbye Sanyang. I would also like to thank my kids for always putting smiles on my face anytime i close from university, especially during stress and difficult times. Secondly, I like to thank my office mates and friends i met here at the university of Birmingham for making our work environment friendly and conducive and also taking their precious times to help me when I need them. Many thanks to Islamic Development Bank (IDB) for giving me the opportunity to undergo this program at the University of Birmingham and generously sponsoring the whole program.

Furthermore, I would also like to humbly put on record my mammoth thanks and appreciation to my learned and able supervisor, Dr. Ata Kaban, for her constant guidance, patience, and yet unspoken assistance during the course of this thesis. Her constructive criticism, encouragements and meticulous checks on my work have always been a source of inspiration for me. I am also indebted to my research monitoring group members, Dr. Steve Vickers and Professor Peter Tino for their invaluable comments, suggestions and monitoring of my progress throughout this program.

I would also like to thank Dr. Robert (Bob) Durrant for his helpful discussions and collaboration/joint paper we have from the work of chapter 4 of this thesis.

Huge thanks are also due to my examiners Professor John McCall and Dr. Shan He for taking time out of their busy schedules to examine my thesis and providing me with valuable comments and suggestions. The same thank is due also to Dr. Rami Bahsoon for chairing the Viva.

Finally, I would like to express my profound gratitude to my parents for their support, prayers and patience for all these years of my program.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Context of the work

The field of optimisation is very important and it has applications in lots of disciplines. In our day to day life, we try to optimise things. For example, if we want to move from one destination to another, we find the shortest path. If we want to invest money, we look for those instruments with minimal risks and high gains. Therefore, the process of optimisation can be thought of as making best use of resources under given constraints. Optimisation can be defined as the procedure or procedures used to make a system or design as effective or as functional as possible. Optimisation helps us to improve quality of decision making. It has applications in Engineering, Business, Economics, Science, Military planning, just to name a few. We may not take it as an exaggeration if one says that in everything we do, we try to optimise.

Evolutionary computation which is a subfield of artificial intelligence refers to a group of algorithms inspired from the Darwinian theory of natural evolution. Methods such as genetic algorithms, evolutionary strategies, particle swarm optimisation, and so on, apply concepts similar to those controlling biological organisms, such as selection and reproduction to solve computing problems. Considering their triumph at finding the global optimum of intricate objective functions, with a good solution quality, evolutionary techniques are often seen as optimisation tools. Evolutionary algorithms (EAs) change a population of candidate solutions over time to a given optimization problem using two operators: selection and variation. Selection introduces a pressure directed toward very good solutions, whereas variation makes sure there is an exploration of the search space of all likely good solutions. Two variation operators are customary in evolutionary and genetic computation: (1) crossover and (2) mutation. The former creates new candidate solutions by combining bits and pieces of promising solutions, whereas the later introduces slight perturbations to promising solutions for exploration of immediate neighbours.

There are different categories of optimisations such as Mathematical Optimisation and heuristic optimisation. Mathematical optimisation only deals with very specific problem types, while on the other hand search heuristics like evolutionary computation work in a black box manner [73]. The latter do not have the guarantees that the mathematical optimisation has, but are less specialised and often work sufficiently well on a wider range of functions. In this research, we are concerned with a heuristic optimisation approach known as Estimation of Distribution Algorithms (EDAs). EDAs build a probabilistic model of fittest individuals and sample from the built model to generate new individuals with the hope of getting improved solutions (See Chapter 2 for more detailed explanation of EDAs). We focus on continuous valued search spaces. We will study a specially commonly used type of unconstrained optimization problem called the box constraint optimisation because it is important, relevant in practice and far from being solved as evident in the annual competitions, such as the one held at Congress of Evolutionary Computation (CEC).

We shall be looking at only minimization problems since maximization problems can be changed to minimization problems by negating the signs of their objective functions. Formally, the mathematical formulation of an optimisation problem for a box constrained type is of the form:

$$\begin{aligned} &\min_{x \in S} \quad f(x) \\ &\text{where} \qquad S \subset R^n, \;\; S = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_n, b_n]. \end{aligned} \tag{1.1}$$

$x \in S$, S is called search space, $a_1 < b_1$, $a_2 < b_2 \ldots$ are given constants and are real values. f is the objective function we want to minimise.

Heuristic search techniques are stochastic optimisation strategies that hope to make candidate solutions better at each iteration, using a fitness function as a guide. Most of the current techniques don′t do well when the problems involve interacting variables and at high dimensions. Presently there are techniques that are able to solve some of the problems encountered in large scale continuous optimisation including problems of dimension of up to 1000. Although successes have been registered in some of the recent techniques, there are still room for improvements in some of them as you will see shortly.

1.2 Motivation

The manner in which evolutionary computation methods like Evolutionary Algorithms (EAs) work depends on operators such as, cross over and mutation and various parameters. Here, the operators of crossing and mutating don't guarantee the preservation of the building blocks. One remedy to the above problems associated with GAs is to replace the traditional genetic operators by building a model of selected good solutions, and generating a new population of candidate solutions using the model with the expectation of improved solution quality. When the built model is a probability distribution, such type of evolutionary algorithms go by the name of Estimation of Distribution Algorithms (EDAs).

EDA addresses a broad class of optimization problems via the learning of explicit probabilistic models of promising candidate solutions and sampling from the built models to generate new candidate solutions. By taking in two concepts of Genetic and Evolutionary Algorithm, like population based search, and exploration through combining good solutions and one from machine learning such as the Probabilistic model estimation, EDA can explore a lot of regions in the search space and enables the use of extremely thorough and careful statistical modeling and sampling techniques to discover and exploit regularities for better exploration. By these incorporations, EDA can solve many challenging problems and can perform significantly better than standard Genetic, Evolutionary and other optimisation techniques.

We are interested in EDA because it has features that other Evolutionary Algorithms (EAs) don't have. For example, it builds a probabilistic model of the search space which provides a better understanding of the problem domain. Higher modelling cost may bring benefit of fewer evaluations or better solution quality. EDA achieves a better and more rigorous theoretical analsysis of the evolutionary process than the GA [49]. EDA's parameters can also be analyzed to understand the structure of the problem. There are lesser and easier parameters to be set in EDAs than in the GAs. With a rigorous analysis of these parameters, we can improve the ability of EDA to efficiently guide the search for the optimum. EDA is chosen in this research due to its increasing applications to real life problems. We can apply EDA to reconstruct images in image recognition. We can use it in Gene expression analysis and in high dimensional data analysis. EDA is known to be remarkably successful in low dimensional problems but prone to the curse of dimensionality in larger problems. There are a number of factors that are responsible for EDAs bad performances in large scale problems, namely the exponential growth of the search space of a problem as the number of decision variables increase, the change in the properties of the search space as the dimensionality of the problem increases (e.g the case of rosenbrock) [76], the high cost of evaluating large scale problems as typical of many real world problems [51],[43], [53] and the interactions between decision variables. Since the main difference between EDA and other Evolutionary and Genetic Algorithms lies in the model estimation of the fittest individuals, the main and most notable challenge for further research in EDA area lies in the model estimation in EDA for high dimensional problems [46],[22]

In this thesis, we tasked ourselves to mitigate the curse of dimensionality associated with EDA in order to scale it up to large scale problems. In particular, we shall be making use of techniques that we will borrow from random matrix theory to scale some EDA up to large scale problems. In so doing, we will make EDA better explore the high dimensional search space with a balance of exploration and exploitation. We shall also utilise the notion of effective dimension of certain class of problems with special structure to remedy EDA susceptibility to the curse of dimensionality.

1.3 Research Questions

In most EDA methods, Gaussian distribution is used as the search operator to provide a probabilistic model of the fittest individuals. However, research has established that Gaussian EDA is prone to premature convergence when its parameters are estimated using the maximum likelihood estimation (MLE) method [49]. One of the promising lines of work in addressing the premature convergence associated to Gaussian was the replacement of Gaussian distribution with a Cauchy distribution in a univariate setting [86]. In this way, the methods will be able to escape the premature convergences due to the ability of Cauchy to make long jumps, thus achieving a better performance. However, some researchers found Cauchy not doing well in high dimensions in comparison with Gaussian. Therefore, we would like to know the answers to the following questions during the course of this PhD work:

1. (a) Will the finding in the previous work of replacing Gaussian search distribution with Cauchy search distribution in univariate setting hold in the multivariate case?
   (b) Can we resolve the controversy surrounding the use of Cauchy search distribution as an alternative search distribution to Gaussian, particularly in high dimensional problems?

A recent method scaled up EDA to high dimensions by employing a technique borrowed from random matrix theory called random projection in an ensemble manner. This method is called Guassian random projection ensemble EDA (RP-Ens-EDA). In this method, there is a random projection matrix parameter whose entries are drawn i.i.d from a Gaussian distribution to project the points down to low dimensions. We would like to extend this method to generate the entries of the random projection matrices from a heavy-tailed distribution again instead of the commonly used Gaussian or sub-Gaussian to strike a balance between the exploration and exploitation of the search process by controlling the size of the covariance matrix. Therefore, we would like to know the answer to the following question during the course of this PhD work:

2. Can we improve the performance of Gaussian Random Projection Ensemble EDA (RP-Ens-EDA) by employing heavy tailed random matrices to control the size of the covariance in order to balance exploration and exploitation of the search process?

Large scale optimisation is very challenging and it limits the usefulness of problems in practice. However, there are problems that are high dimensional but can possibly be represented in low dimension [18]. Also It has been noted from research that in certain classes of functions most decision variables don't impact significantly on the objective function. Such functions are said to have low *intrinsic dimensionality*. We want to utilise the structure of functions with intrinsic dimension by employing random embedding technique to scale up Estimation of Distribution Algorithms (EDA) for problems with low intrinsic dimension. Therefore, we would like to know the answer to the following question during the course of this PhD work:

3. Can we devise a method that exploits intrinsic dimension without knowing the influential subspace of the input space, or its dimension, by employing the idea of random embedding to mitigate the curse of dimensionality associated with EDA?

1.4 Contributions

The main contributions of this thesis are summarised as follows:

1. We were able to extend the promise of using Cauchy search distribution in univariate EDA to multivariate setting in chapters 3 and 4.

2. We have also resolved the controversy around the use of Cauchy search distribution and its merits relative to the Gaussian, in high dimensional problems in chapter 4 since employing heavy tailed search distribution, such as a Cauchy is found to make EDA better explore a high dimensional search space while also being found that Cauchy search distributions is less effective than Gaussian search distributions in high dimensional problems.

3. In chapter 5, we extend a recently proposed random projections (RP) ensemble based approach by employing heavy tailed random matrices, which achieved a flexible means of balancing exploration and exploitation of the search process.

4. In chapter 6, we used random embedding technique to remedy the curse of dimensionality associated with the plain EDA specifically when intrinsic dimension of the problem is low.

1.5 Publications

During the course of this PhD work, the following peer-reviewed papers have been published in conference proceedings and workshops. The following technical report was also written:

Published Refereed Conference Papers

- M.L. Sanyang and A. Kaban. Multivariate Cauchy EDA Optimisation. In proceeding of the Intelligent Data Engineering and Automated Learning (**IDEAL**-2014), pp. 449-456. (2014).

- M.L. Sanyang and A. Kaban. Heavy Tails with Parameter Adaptation in Random Projection based Continuous EDA. In proceeding of the IEEE Congress on Evolutionary Computation, (**CEC**-2015), pp 2074-2081, IEEE (2015). **Runner-Up Best Student Paper Award**.

- M.L. Sanyang, R.J. Durrant and A. Kaban. How effective is Cauchy-EDA in high dimensions? In proceeding of the IEEE Congress on Evolutionary Computation (**CEC**-2016), 24-29 July, Vancouver, Canada, (2016). **Nominated by a reviewer for a best paper award**.

- Qi Xu, M.L. Sanyang and A.Kaban. Large Scale Continuous EDA Using Mutual Information. In proceeding of the IEEE Congress on Evolutionary Computation (**CEC**-2016), 24-29 July, Vancouver, Canada, (2016).

- M.L. Sanyang and A. Kaban. REMEDA: Random Embedding EDA for optimising functions with intrinsic dimension. In proceeding of the 14-th International Conference on Parallel Problem Solving from Nature (**PPSN** XIV), 17-21 September, Edinburgh, Scotland, (2016). **Nominated for Best Paper Award**.

Published Refereed Workshop Paper

- M.L. Sanyang, Hanno Muehlbrandt and A. Kaban. Two approaches of Using Heavy Tails in High dimensional EDA. In proceedings of the International Conference of Data Mining Workshop, (**ICDMW**-2014), (2014).

Technical Report

- M.L. Sanyang and A. Kaban. How effective is Cauchy-EDA in high dimensions? Technical Report **No. CSR-15-01**, School of Computer Science, The University of Birmingham.

1.6 Thesis Outline

This thesis consists of 7 chapters which are organised as follows. Chapter 2 presents the essential background, basic concepts of EDA and existing work in the area of high dimensional continuous EDA methods.

Chapter 3 presents the extension of the use of Cauchy distribution as an alternative search distribution to blend together the advantages of multivariate modelling with the ability of escaping early convergence to efficiently explore the search space.

Chapter 4 focuses on resolving the controversy around the merits of using Cauchy search distribution as opposed to the use of Gaussian search distribution in optimization, particularly in EDA optimization.

Chapter 5 proposes an extension of a recently Random Projection Ensemble EDA method by generating the entries of the Random Projection Matrices I.I.d from a heavy tailed distribution (t-Distribution in this case) instead of generating the entries from the commonly used Gaussian or sub-Gaussian. The use of t-distribution may look surprising from the perspective of random projections; however we show that the resulting high dimensional ensemble covariance is enlarged when the degree of freedom parameter is lowered, which may facilitate exploration and escape early convergence, while still maintaining the focus of the search.

Chapter 6 presents a framework that utilizes the random embedding technique in Estimation of Distribution Algorithm to scale it up by exploiting the intrinsic dimension of problems whereby the search takes place in a much lower dimensional space than that of the original problem. This framework is suited for large scale problems that take a large number of inputs but only depend on a few linear combinations of them.

Chapter 7 summarises the main findings of this thesis and gives conclusions of the proposed work with outlooks for future work.

CHAPTER 2

Basic concepts, background and review of heuristic Optimisation

2.1 Notations and their descriptions

D	The Ambient dimension of problems.
N	Population Size.
θ	Correlation threshold that divided weak and strongly dependent variables.
$\tilde{N}$	Number of selected/fittest individuals.
$runs$	Number of runs.
$MaxBudget$	Maximum number of function evaluations.
f	Fitness function.

X	Represents individuals/Decision vector.
S	Represents the search space.
g	Denotes generation/iteration.
P	Represents the populations of individuals.
P^{Sel}	Set of Selected/fittest individuals.
P^{new}	Newly sampled population.
R	Random projection Matrix
A	An $N \times D$ Matrix.
y	A point/newly sampled individual.
x	An element of the decision vector, X.
Σ	Covariance Matrix.
μ	Mean.
σ	Standard deviation.
$\mathcal{M}$	Represents a probabilistic model.
X_{new}	Newly sampled vector.
d_i	Denotes the intrinsic dimension of problem.
d	Denotes the internal dimension, such that $D > d > d_i$.

2.2 Heuristic Optimization in Continuous Domains

This section gives an introduction to a handful of methods currently applied to Large Scale Global Optimisation (LSGO) problems, which are related to our work. Readers interested in more detailed surveys are referred to [39],[54] and [61].

Heuristic methods, sometimes called "approximate" methods, are techniques coined for providing approximate solutions to hard search or optimisation problems for which there is no specialised algorithm, i.e problems that would not be solvable in other ways even approximately.

The goal of a heuristic is to give rise to quick enough a solution that is sufficient for solving the problem at hand. This solution may not be the best or it may simply approximate the exact solution to the problem at hand, but it is still valuable because finding it does not require a prohibitively long time. Just by themselves alone, heuristics may produce results, or they may be used in association with optimization algorithms to improve their efficiency. For example they may be used to generate good seed values. An example of heuristic algorithm that we intend to utilise in this research is Estimation of Distribution Algorithm. EDA is presently prohibitively costly at high dimensions, so we shall be using techniques such as random projection to improve its performance at high dimension.

2.2.1 Evolutionary Algorithms (EA)

Evolutionary algorithms are population based heuristic search techniques, used for several decades to find approximate solutions to optimization problems. The classical EA alternates two phases: (i) information exchange among individuals by evolutionary operators, and (ii) fitness evaluation and selection [33]. Most of the time, EAs employ cross over (also known as recombination) and mutation as evolutionary operators. The main essence of cross over is the exchange of genetic material among individuals, while mutation allows exploration of areas close to an existing candidate solution [54]. The pseudo code of an EA implementation is shown in Algorithm 1.

Generally, EAs are unable to detect inter-dependencies among the search variables. In particular, the cross over operator tends to break up good building blocks, which decreases significantly the performance of EAs on certain problems [54]. This is where Estimation of Distribution Algoritms (EDA) offer advantages.

Algorithm 1 Classical Evolutionary Algorithm

```
Set $g \leftarrow 0$.
Set $P_g \leftarrow$ initialize random population.
**While** Stopping Criteria is not met **do**
        $F \leftarrow$ evaluate fitness of $P_g$
        $P_g^{'} \leftarrow$ Perform crossover
        $P_g^{''} \leftarrow$ Perform mutation on $P_g^{'}$
        $P_{g+1} \leftarrow$ merge $P_g$ and $P_g^{''}$
        $g \leftarrow g + 1$
**end**
**Output:** $P_{g+1}$
```

2.2.2 Estimation of Distribution Algorithms (EDAs)

EDAs represent a branch of EAs that replace the cross over and mutation operators by building and sampling explicit probability models of selected candidate solutions [49]. This allows the algorithm to learn the structure of the search space and to guide the search in further promising directions [87]. We can say they are a variation of regular EAs.

2.3 Introduction to Estimation of Distribution Algorithms (EDA)

The major difference between EAs and EDAs is the replacement of the recombination and mutation operators by a probabilistic model. A typical EDA algorithm proceeds by initially generating a population of individuals of size N. At each generation g, the current population P_g is evaluated using the objective function. Based on the fitness evaluation, the most promising individuals are selected according to their fitness values to build a probabilistic model and sample the next generation of candidate solutions from this model. A popular selection scheme used by many EDA variants is truncation selection. Truncation selection simply defines a percentage of individuals τ to be selected out of the entire population of size N and selects the subset P^{sel} of candidates with the best fitness values by choosing the top $\tau \times N$ individuals of the population. The τ controls the selection pressure. Figure 2.1 illustrates the procedure of a simple EDA implementation and the pseudocode of a simple EDA can be found in Algorithm 2, which is adopted from [73].

Algorithm 2 The Pseudocode of a simple EDA with Population size N

```
(1) Set $g \leftarrow 0$.
(2) Set $P_g \leftarrow$ Generate $N$ points uniformly randomly to give an initial population.
Do
    (3) Evaluate fitness for all $N$ points in $P_g$
    (4) Select the $\tau \times N$ individuals $P^{sel}$ from $P_g$
    (5) Calculate the sample statistics of $P^{sel}$
    (6) Use the statistic to sample new population $P^{new}$
    (7) $P \leftarrow P^{new}$
    (8) $g \leftarrow g + 1$
Until Termination criteria are met
Output: $P$
```

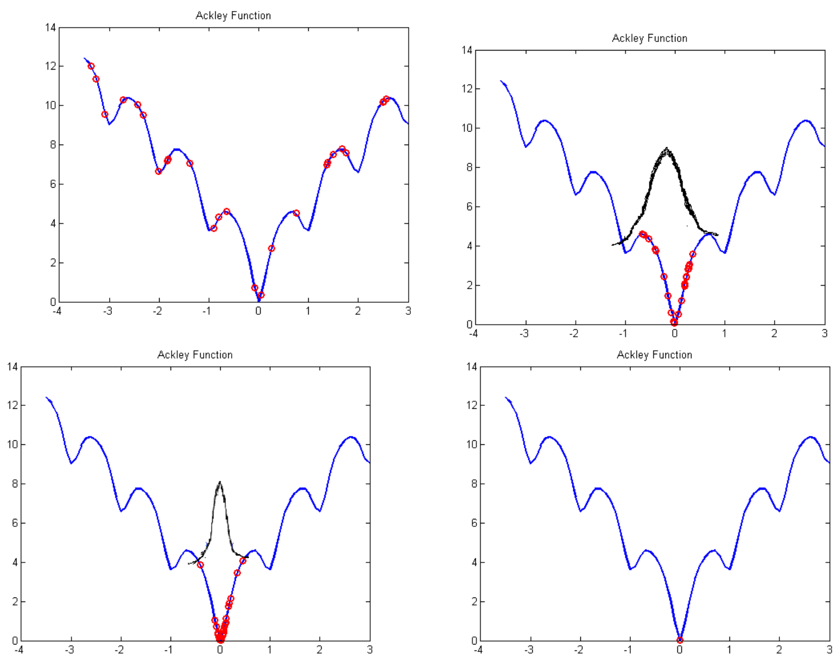


Figure 2.1: Evolutionary procedure of a simple EDA on one dimensional problem

In figure 2.1, the top left graph depicts the situation after uniformly initializing the population in the search space. The x-axis/horizontal axis is the search space and the y-axis/vertical axis is the fitness of the candidate solutions. After the first generation, the probabilistic model is estimated from the selected population P^{sel}. In the top right and bottom left graphs, the probability model is indicated by the bold black bell curve. In the bottom right graph, the population has converged close to the global optimum.

2.3.1 Model estimation in EDAs

The basic skeleton of most EDA approaches is very similar to the one described in Algorithm 2, But the cream of each EDA variant is the process of building its model. The probabilistic model has a fundamental role and influence on the performance of the algorithm, since it is most of the times the only tool used for guiding the search to the global optimum [56]. Thus, the model must model the search space in great detail, meaning, if the search space involves search variables which interact which each other, then the model has to be a multivariate model that takes into account these interactions. However, the required search cost (i.e. number of fitness evaluations used) and computational resources have to be appropriate as well. This becomes a problem when the search space is high dimensional, and therefore simplified models such as univariate models are frequently used instead of full multivariate models. This means that all decision variables, x which is a coordinate of the vector X, are treated as if they were independent throughout the model building process, and for each decision variable a separate model is estimated. An early approach is the $UMDA_c$, which is described in [61]. The model building in this univariate approach is done using Equation 2.1 [61].

$$P(X) = \prod_{i=1}^{D} P(x_i) \tag{2.1}$$

Where X is a set of decision variables and x_i are instances of these decision variables.

D is the Ambient dimension of the problem. For problems that are not separable (see definition in section 2.5.3), the design variables have inter-dependencies. One of the first approaches to capture those inter-dependencies has been the MIMIC algorithm, which was proposed by *De Bonet et. al* in [42]. This model has the potential of capturing bivariate interactions between decision variables by sampling from the pairwise joint distribution between variables according to Equation 2.2. Given a permutation $pm = (i_1, i_2, ..., i_D)$, the class of probability functions $P_{pm}(X)$, is defined as

$$P_{pm}(X) = P(x_{i_1}|x_{i_2}) \cdot P(x_{i_2}|x_{i_3}) \cdots P(x_{i_{D-1}}|x_{i_D}) \tag{2.2}$$

Despite MIMIC is able to outperform univariate models, the majority of optimization problems will have larger groups of interacting design variables. Several proposals have advanced to explicitly capture multivariate dependencies by building graphical dependency networks, for example Bayesian Networks [11] but, from statistical, computational and memory points of view, learning probabilistic graphical models is highly expensive [6]. Thus, the scaling up of these model building processes to high dimensional problems is challenging. Efforts to mitigate such problems have resulted in several algorithms being proposed recently. For example, an approach that is correlation based such as in [22] and a variable interaction learning process such as in [83] were such proposals. Both of these methods will be discussed in detail later.

2.3.2 The difficulties of model estimation on high dimensional EDA problems

EDAs are the class of Evolutionary Algorithm, which replace cross-over and mutation operators with the building of probability models from selected individuals. In so doing, they completely rely on probabilistic models built from selected individuals. They therefore suffer from the well-known curse of dimensionality [34]. The more complicated the model is, the more individuals it needs to give a reliable estimate and maintain good performance. The curse of dimensionality implies that, the amount of data to maintain a given spatial density increases exponentially with the dimensionality of the search space [8]. Since we can see from Algorithm 2 that EDA tries to learn some global statistical details from P^{sel} individuals selected from the population P of N individuals, P^{sel} has to be large enough for reliable estimation, which will consequently require a large population size N for some level of selection pressure to be maintained. Therefore, the population size of EDA has to grow fast as the problem size grows to maintain good performance. Thus, we can say that there is an estimation issue in EDA as the sample size required to produce a reliable estimate of the distribution of selected individuals grows exponentially with the dimension of the search space [78].

Apart from the estimation problem EDAs have, multivariate EDA applications to large scale problems are rare due to their high computational requirement. As already mentioned, high dimensionality comes with the need for a large number of fitness evaluations implied by the need for large population size. Even that aside, the computational complexity of estimating the model and sampling new individuals using the estimated model in a full multivariate EDA is also significantly higher than that of the search operators in other EAs. The computational cost of sampling from a full d-dimensional Gaussian distribution is $O(D^3)$ [24], because we have to do an eigen decomposition of the Covariance matrix and this takes $O(D^3)$ in general. This process becomes extortionate when D is very large.

When the problem dimensionality increases, we see an exponential increase in the size of the solution space. For instance, when the problem dimensionality increases from 5 to 20 in a binary optimization problem, the size of the solution space increases from $2^5 = 32$ to $2^{20} = 1048576$ exponentially in the dimensionality. The speedy growth in the size of the search space makes it extremely strenuous to find the global optimum by sampling/searching the entire space. However, in certain type of functions, the function value changes a lot along certain directions and along other directions, the function value does not change much or not at all. This is a typical case of someone tuning several parameters and when he/she changes some of them, not many changes are notices in the performance, but when he/she changes the others, we notice big difference in performance. The directions/dimensions that make the huge change in the function value are called the *intrinsic dimensions*, while the dimensions that make little or no change in the function value are called the *constant dimensions*. Therefore, to address the scalability issue in optimisation, one can consider only the dimensions that impact significantly on the function value in the optimisation process, if the functions are of this form. Thus, instead of searching in the ambient dimensional space, if you search along the intrinsic dimensional sub-space, which is much lower than the ambient space, you should be able to find the optimum. In this way, we can solve large scale black-box optimization problems easier.

2.4 Random Projection

Now we will describe a technique of universal dimension reduction, that originates from theoretical computer science, which made some recent advances in Machine Learning, and is very new to optimisation. When points are in high dimension, random projection method can reduce the dimension of the data to a lower one in such a way that the pairwise distances between two points of the dataset are preserved with high probability. Our research will make use of random projection technique to solve some of our research questions. Specifically we will extend and improve the ensemble of random projections technique recently proposed in [46].

Here we will discuss this method, particularly the ensemble of Random Projections. Among the challenges we highlighted in the previous section is the curse of dimensionality. This is one of the challenges faced by researchers in heuristic optimisation, especially by

Estimation of distribution Algorithms (EDAs) for large scale global optimisation. There are typically two problems that motivated researchers to endeavour to mitigate this curse of dimensionality [26]:

1. Very high dimensional data and very many observations. Computational time and space complexity are the problems here.

2. Very high dimensional data and very few observations. This represents a hard problem in inferencing and results in fake interaction estimates between features [26].

The curse of dimensionality becomes a problem when trying to solve optimisation problems. The objective function must be computed for each combination of values. This creates an obstacle when the dimension of the state variables is large. This problem also prevails in other desciplines such as machine learning, combinatorics, Bayesian statistics etc. Due to this problem, researchers try to solve this issue by reducing the dimensionality, which can be achieved using either Principal Component Analysis (PCA), Factor Analysis (FA), Random Projection (RP) etc. PCA was used in one of Dong's [23] papers, but the computational complexity is still $O(D^3)$. Random Projection was used more recently and demonstrated a drastic improvement, so we choose to build on this.

The most widely theoretical motivation behind the use of random projections is the Johnson-Lindenstrauss lemma, which is stated as follows [52]. The distributional Johnson-Lindenstrauss lemma (JLL) [19],[4] is state as follows:

Lemma 1. Let $\varepsilon, \delta \in (0, 1)$. Let $k \in \mathbb{N}$ such that $k \in \mathcal{O}(\varepsilon^{-2} \log \delta^{-1})$. Then there is a random linear mapping $R : \mathbb{R}^D \to \mathbb{R}^k$ such that for any vector $X \in R^D$, with probability at least $1 - \delta$ it holds that:

$$(1-\varepsilon)||X||_D^2 \leqslant ||RX||_k^2 \leqslant (1+\varepsilon)||X||_D^2 \qquad \square$$

There are different proofs of this lemma such as [52] and [4]. For readers interested in the proof, you can check [19] for the elementary proof.

Projecting from D dimensions to k dimensions is a linear transformation represented by a $D \times k$ matrix R, which is generated by drawing each entry of the matrix from an $i.i.d$ $N(0, 1/k)$ distribution. In the worst case, k has to be $\mathcal{O}(\log N/\varepsilon^2)$. Given a D-dimensional data set which is represented as an $n \times D$ matrix A, where n is the number of instances in A, the mapping $A \times R$ results in a reduced dimension.

The entries of the R matrix are normally generated $i.i.d$ from a Gaussian, which have the advantage of making the matrix full row rank a.s [46]. However, there are several other choices available in the literature, which are faster when doing matrix-multiplication, and still have full rank with very high probability. Among the other choices, we will mention two here, namely a sparse Random Projection matrix $(R_{i,j})$ proposed in [3] that has entries drawn $i.i.d.$ as following:

$$R_{i,j} = \begin{cases} +\sqrt{3} & with\ probability\ \frac{1}{6}, \\ -\sqrt{3} & with\ probability\ \frac{1}{6}, \\ 0 & with\ probability\ \frac{2}{3}. \end{cases} \tag{2.3}$$

and the Random Projection matrix $(R_{i,j})$ with coin-flip entries [3]. This is also use due to its computational efficiency and has its entries drawn according to the following:

$$R_{i,j} = \begin{cases} +1 & with\ probability\ \frac{1}{2}, \\ -1 & with\ probability\ \frac{1}{2}, \end{cases} \tag{2.4}$$

2.5 Random Embedding

Given a random embedding matrix $R \in \mathbb{R}^{D \times d}$ and points $y_i \in \mathbb{R}^d, i = 1, \cdots N$, where N are the number of points and $d < D$, the operation Ry_i transforms the d-dimensional points into the D-dimensional space of decision variables. This transformation is what we call random embedding. The matrix R will have its entries drawn i.i.d. from a standard Gaussian or subgaussian distribution.

2.5.1 How to create a non-separable functions from a separable one.

In an attempt to build a non-separable problem from a separable one, we rotate the coordinate system as follows:

$$f : x \longrightarrow f(x) \quad separable \tag{2.5}$$

$$f : x \longrightarrow f(Rot \cdot x) \quad non-separable \tag{2.6}$$

Where *Rot* is a rotation matrix

We shall be using non-separable functions in our subsequent chapters, so it will be good to show how it can be obtained from separable functions here.

2.6 Definition of type of benchmarking functions used in this thesis

In this section, we will give definitions of the type of benchmark functions used in this work. For each function, a graphical representation and detailed analysis of their characteristics are provided in the appendix. We have used two types of benchmark functions, namely the CEC 2005 and CEC 2010, which are described in [77] and [1] respectively. The functions have been slightly modified and expanded to accommodate the needs of our experiments.

2.6.1 Separable Functions

A variable x_i on a function is *separable* or does not interact with any other variable if the following holds:

$$\arg\min_{X} f(X) = \left(\arg\min_{x_i} f(X), \arg\min_{\forall x_{j,j \neq i}} f(X) \right) \tag{2.7}$$

Where $X = (x_1, ..., x_D)$ is a decision vector of D dimensions. The notation $\arg\min_{x_i} f(X)$ means that the optimum value of x_i is found while all other variables are kept constant [59].

In other words, for *separable* functions, each dimension can be considered as independent group and the global optimum can be found by optimising in each of its arguments separately.

2.6.2 Partially Separable Functions

If there exist at most $m < D$ disjoint subsets $u_1, u_2, ..., u_m \subset X$, where $X = (x_1, ..., x_D)$ is a decision vector of D dimensions, then a function is *partially separable* iff:

$$\arg\min_{X} f(X) = \left(\arg\min_{x_1} u_1(x_1, ...), ..., \arg\min_{x_m} u_m(..., x_m) \right) \tag{2.8}$$

A *non-separable function*, which is a function that has atleast one variable that interacts with any other variable, is called *m-non-separable* if at most m of its arguments are not independent.

2.6.3 Fully non-separable Functions

A function $f(X)$ is said to be *fully non-separable* if every pair of its decision variables X interact with each other. When pairs of variables are not independent or are correlated, then there is interaction between the variables. This is what variable interaction means.

2.6.4 Partially additively separable Functions

A function is partially additively separable if it can be expressed in the following form:

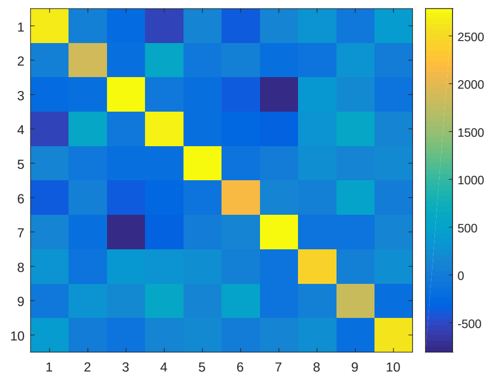
$$f(X) = \Sigma_{i=1}^{m} f_i(x_i) \tag{2.9}$$

where x_i are disjoint decision vectors of f_i and $X = (x_1, ..., x_D)$ is a global decision vector of D dimensions and m is the number of independent subcomponents [59],[55].

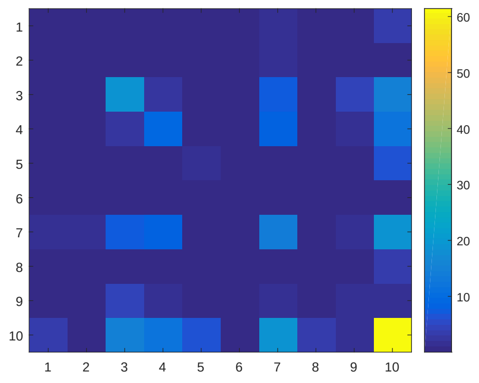
Experiments of this thesis talk about benchmark functions which are separable and non-separable. Due to their usage in the thesis, we have given their definitions above.

2.6.5 EDA captures structures of functions

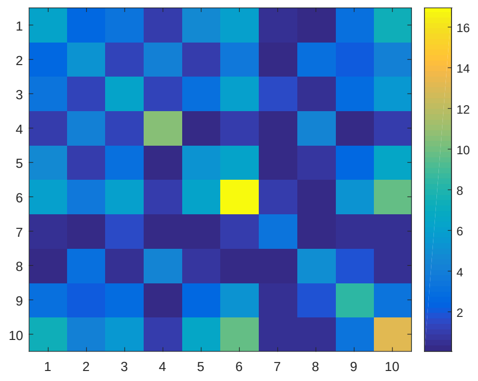
To demonstrate the ability of EDA to capture the structure of the problem on figure 2.2, we show examples of covariance matrices of the set of selected fittest individuals extracted from a typical run. The covariances of the figure are for three different problem types: Separable, non-separable and expanded functions. As you can see from the image of the covariance matrix of the sphere function, Figure 2.2 (a), it shows a diagonal covariance which indicates that the problem is a separable one. From figures 2.2 (b) and (c), we can see that the rosenbrock and expanded functions are non-separable as the off diagonals indicate that there are interactions between the decision variables.



(a) Sphere function



(b) Rosenbrock function



(c) Expanded function

Figure 2.2: Figure showing captured structure from the CEC 2005 benchmarks

2.7 Differences between multivariate Gaussian and multivariate Cauchy distributions

The main advantage of EDAs is the explicit learning of the dependences among variables of the problem to be tackled and utilizing this information efficiently to generate new individuals to drive the search to the global optimum [87]. Using univariate Cauchy will make it hard to achieve this goal since it does not take on dependences. A univariate distribution is a distribution that takes only one variable as its argument, while a multivariate distribution takes more than one variable. When a univariate distribution is used on a multivariate problem, It treats each variable independently from the others, while a multivariate distribution takes the dependencies between the variables into consideration. Therefore, this work to the best of our knowledge is the first to include the modelling of dependencies in a Cauchy search distribution based EDA algorithm for black-box global Optimization.

An important difference between Gaussian and Cauchy is that Cauchy is heavier tailed. This means that it is more prone to producing values that fall far from its mean, thus, giving Cauchy more chance of sampling further down its tail than the Gaussian. This gives Cauchy a higher chance of escaping premature convergence than the Gaussian [69], [86].

For the same reason, the Gaussian search distribution is good when the individuals are close to the optimum while Cauchy is better when the individuals are far from the optimum. Both of these findings were previously made in the context of traditional evolutionary computation [86] where univariate version of these distributions were used to implement the mutation operator. Our hypothesis, which we test in this chapter, is that these advantages are carried forward to EDA based optimization where in addition, multivariate modelling enables a more directed search. Below is a picture that shows the plots of both Gaussian and Cauchy Distributions in one and two dimensions:

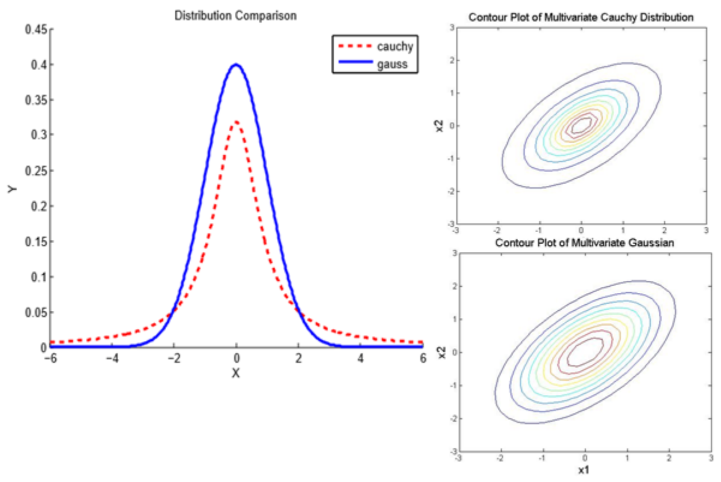


Figure 2.3: Cauchy density (red dashed), along with standard normal density (blue) (Left), multivariate Cauchy (Top Right) and multivariate Gaussian (Bottom Right).

The leftmost plot in figure 2.3 shows the probability density function of a Cauchy versus a Gaussian in 1D. We see the heavy tail of Cauchy falling down slower than Gaussian. On the right, the plots depict contour plots of the 2D versions of these densities, with Cauchy on the top right and Gaussian at the bottom right. In the 2D versions, Cauchy has a flatter tail on the base as can be seen by the wider space between the second outermost and the outermost contour lines than those of Gaussian. Parameter $\Sigma = [1.6; .61]$ was used on both 2D version for plotting the contours.

2.8 Review of previous work on high dimensional EDA methods

Large scale continuous optimization has put lots of interest in most of the researchers in classical and other heuristic methods for continuous optimization problems, due to its appearance in real-world problems. Most of the problems are high dimensional. Therefore, scaling up for high-dimensional problems becomes a challenge, which makes model building in high dimensions a subject of many research efforts. Many approaches were proposed, here we will limit ourselves to a few most relevant ones. We will focus on continuous box unconstrained optimisation problems and start with an early approach called Continuous Univariate Marginal Distribution Algorithm ($UMDAc$), whose model bulding is kept very simple. Amoung the other methods to be reviewed here are Eigendecomposition EDA ($ED - EDA$) [24], which proposes to utilise a repaired version of the full covariance matrix estimate, thus giving it the ability to capture interaction among decision variables. For variables to interact, equation 2.7 will not hold. This is a concept that is related to independence. Other methods use limited dependencies. For example, Cooperative Co-evolution with Variable Interaction Learning ($CCVIL$) proposed by *Weicker et al.* in [83] is a deterministic method to uncover dependencies between decision variables, which has later been extended to the CCVIL framework by *Chen et al* in [80]. EDA with Model Complexity Control ($EDA - MCC$) [22] also employs a deterministic algorithm to group variables. It splits all decision variables into two independent subsets, one set contains decision variables with only minor interaction with other variables and the other contains strongly dependent variables. Other methods are Covariance Matrix Adaptation *(CMA-ES)* [36], separable CMA-ES (*sep-CMA-ES*) [70] Multilevel Cooperative Co-evolution ($MLCC$)[88] and Random Projections Ensemble EDA (rp-Ens-EDA)[47]. In the following sections, we will discuss each of these methods.

2.8.1 Continuous Univariate Marginal Distribution Algorithm (UMDAc)

The UMDAc is an early approach of a univariate EDA for black-box optimisation in continuous domains. The model building process of this algorithm is kept quite simple, with the expectation of a reduced computational resources requirement compared to more sophisticated variants. The disadvantage of keeping the technique in this algorithm simple is that it does not take any interaction among the decision variables into account. From a loose definition, Variable interaction is the extend to which the optimization of a decision variable is affected by the values taken by other decision variables. We have also noticed that most current EDA implementations employ a Gaussian distribution for sampling offsprings.

Intuitively, we expect that a univariate EDA will perform well on separable problems, but the simple model building process will cause a significant performance deterioration on problems with large amount of variable interactions. A common problem of EDAs that use Gaussian distribution to sample new individuals is premature convergence. Going by the fact learned from literature, sampling from a Cauchy distribution will prevent premature convergence for a longer time compared to the common sampling method (i.e, using Gaussian) and, thus, will lead to a more steady improvement throughout the evolutionary process. However, a positive impact, if any, will only be realized for problems with large search spaces, as a Gaussian distribution covers a small problem domain.

Description of the Algorithm

UMDAc is a univariate model, therefore it estimates its probabilistic model M for each dimension separately. In the first generation, the population P of size N is initialized uniformly across the search space. In each generation, the mean μ_j and variance σ_j^2 are estimated for each dimension separately. The maximum likelihood estimates of μ_j and σ_j^2 for dimension j for points i are defined by Equations 2.10 and 2.11 respectively.

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{j,i} \tag{2.10}$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{j,i} - \mu_j)^2 \tag{2.11}$$

Based on the above estimated parameters, a new population is sampled from a Gaussian normal distribution, $\mathcal{N}(\mu, \sigma^2)$.

Algorithm 3 The Pseudocode of UMDAc with Population size N

```
(1) Set $g \leftarrow 0$.
(2) Set $P \leftarrow$ Generate $N$ points uniformly randomly to give an initial population.
Do
        (3) Evaluate fitness for all $N$ points in $P$.
        (4) Select some individuals $P^{sel}$ from $P$.
        (5) Estimate the $\mu$ and $\sigma^2$ for each dimension of $P^{sel}$ by maximum likelihood.
        (6) Generate new population $P^{new}$ from $\mathcal{N}(\mu, \sigma^2)$.
        (7) $P \leftarrow P^{new}$.
        (8) $g \leftarrow g + 1$.
Until stopping criteria are met
Output: $P$
```

The standard UMDAc algorithm with a Cauchy distribution

EDAs can converge quickly and the Gaussian distribution is not able to escape this early convergence, once the variance among the population is small [56]. In the context of Evolutionary Algorithms, the Cauchy distribution, which is a heavy tail distribution, has been successfully employed for allowing larger step sizes in the case of the mutation operator [86]. A heavy tailed distribution is a distribution that is liable to produce large values. They have heavier tails than the exponential distribution. The heavier the tail, the larger the probability that you'll get one or more very large values in a sample. The tails of the Cauchy is larger than the Gaussian ones (see Figure 2.3 for the case of both univariate and multivariate), thus, the Cauchy distribution tends to generate larger random numbers than the Gaussian one.

$$f_q(x) = \frac{1}{\pi}\frac{q}{q^2 + x^2} \qquad -\infty < x < \infty \tag{2.12}$$

The Cauchy distribution is defined by Equation 2.12, where $q > 0$ is a scaling parameter [32]. Large values for q increase the probability of generating larger random numbers (see Figure 2.3). The Cauchy distribution does not have mean and variance. As already discussed in the introduction of this section, throughout the run of an Evolutionary Algorithm, the variance of the population decreases, which allows convergence around the global optimum [56]. This behaviour is not seen when employing a Cauchy distribution. Even though the step size can be controlled by q, keeping q constant will have a negative impact on the algorithm's performance: A large value for q will prevent convergence and a small value for q prevents a good exploration of the search space at earlier stages[56].

2.8.2 Eigendecomposition EDA (ED-EDA)

ED-EDA is a conceptually simple multivariate EDA with some similarities with UMDAc. One can say that UMDAc samples new individuals from a diagonal covariance matrix, thus making it to take the variance of each decision variable into account, but no covariance among different variables. In contrast, ED-EDA samples individuals from a full covariance matrix, which enables it to capture interactions among decision variables.

One general drawback with this algorithm is the quality of its estimated covariance matrix. With a fixed population size N, the quality of the estimated model, M, decreases when the dimensionality D increases [68]. A less exact estimation of the model is likely to cause a deterioration of the algorithm performance, because it increases the difficulty of capturing the fitness landscape. Authors in [24] have shown that their ED-EDA algorithm is very competitive on low dimensional problems (up to 50d), but they have not shown their approach applicability on high dimensional problems. The model will be less capable on high dimensional problems, as its quality will be decreasing with increasing dimensionality. A performance increase on separable problems is not expected, as these kind of functions can be fully optimised by a univariate approach as well.

Description of the Algorithm

In this section, we will provide a detailed description of the ED-EDA algorithm. Basically, ED-EDA learns the structure of the fitness landscape via the covariance matrix Σ. Even though this approach is very direct, it generally suffers from two problems:

1. The covariance matrix needs to be positive semi-definite as described in detail in [23]. However, due to the limited precision of computers with regard to representing decimal numbers, an estimated covariance matrix may not always fulfil this criterion. This so called ”ill-posed” covariance matrices can cause a break down of the algorithm, as they are likely not to allow sampling of new individuals. This obstacle leads to the need for repairing covariances of this kind.

2. As it has been illustrated by Bosman in [12], the estimated covariance matrix does not cover the whole search space. Generally, when sampling new individuals from Σ directly, the area of the search space, which can be explored by the population is limited and is highly dependent on the initialization of the first generation. The continuous selection pressure will shrink the variance within the population, which will cause a favourable convergence at the end of a run. Empirical results have shown that, the initial population does not cover the whole search space, but only a sub-optimal region, as a results an EDA method cannot escape this region, but rather just converge. To increase the exploration ability of EDAs sampling from Σ, covariance scaling has been introduced.

The following is the basic procedure of ED-EDA:

The first step is to estimate the mean μ and the covariance matrix Σ from the selected population P^{sel}. Then, Σ gets decomposed into eigenvectors and eigenvalues in such a way that Equation 2.13 holds. Vec is a matrix of D eigenvectors according to Equation 2.14 and V is a diagonal matrix with dimensionality $D \times D$ of eigenvalues according to Equation 2.15.

$$\Sigma = Vec \cdot V \cdot Vec^T \tag{2.13}$$

where:

Algorithm 4 The Pseudocode of ED-EDA

```
(1) Set $g \leftarrow 0$.
(2) Set $P \leftarrow$ Generate $N$ points randomly to give an initial population.
Do
        (3) Evaluate fitness for all $N$ points in $P$
        (4) $P^{sel} \leftarrow$ Select the fittest individuals $P^{sel}$ from $P$
        (5) Estimate the $\mu$ and $\Sigma$ of $P^{sel}$
        (6) $(Vec), (V) \leftarrow$ Obtain eigenvectors and eigenvalues by eigendecomposition of
            $\Sigma$
        (7) $V \leftarrow$ Repair Eval. Refer to Alg. 5
        (8) $V \leftarrow$ Tune Eval. Refer to Alg. 6
        (9) $P^{new} \leftarrow$ Sample new individuals according to Equation 2.17
        (10) $P \leftarrow P^{new}$
        (11) $g \leftarrow g + 1$
Until stopping criteria are met
Output: $P$
```

$$Vec = [v_1, v_2, ..., v_D] \tag{2.14}$$

$$V = \begin{pmatrix} \lambda_1 & & & & 0 \\ & \lambda_2 & & & \\ & & \cdot & & \\ & & & \cdot & \\ 0 & & & & \cdot \\ & & & & & \lambda_d \end{pmatrix} \tag{2.15}$$

New individuals cannot be sampled directly from the decomposed covariance matrix, but rather from a triangular matrix H, which can be obtained by Equation 2.16:

$$\Sigma = Vec \cdot V \cdot Vec^T = H \cdot H^T \tag{2.16}$$

The triangular matrix H could be directly obtained from Σ by applying Cholesky decomposition, as long as the covariance matrix is positive semi-definite. But, as described in detail in [24], decomposing Σ firstly into eigenvectors and eigenvalues offers good possibility for repairing and tuning Σ before sampling from H.

After tuning the eigenvalues, one can sample new individuals from triangular matrix H according to Equation 2.17, where μ is a D dimensional vector containing the means of each dimension calculated from the selected population, H is the triangular matrix obtained from Equation 2.16, and $X = (x_1, ..., x_D)$ is a D dimensional vector for which $x_i \overset{iid}{\sim} \mathcal{N}(0,1), \ \forall x_i \in X$

$$X_{new} = \mu + H \cdot X \tag{2.17}$$

Where X_{new} is a newly sampled vector.

Algorithm 5 Repair V

```
$\lambda_{min} \leftarrow$ Smallest eigenvalue of $V$
**If** $\lambda_{min} < 0$ **do**
        **V** $\leftarrow$ V + $\mid \lambda_{min} \mid \cdot I$
**endIf**
```

Algorithm 6 Tune V

```
**for** $i = 1...D$
  **If** $\lambda_i < 0$
      $\lambda_i \leftarrow 0$
  **endIf**
**endfor**
```

Covariance repairing

Here we talk about Covariance repairing[1]. In case the covariance matrix is ill-posed as described already, obtaining triangular matrix H according to Equation 2.16 is impossible. The authors in [24] proposed two techniques, which do not repair Σ directly, but rather

[1]Most of the methods discussed in the following three section are not based on the covariance matrix, but rather on its eigenvalues. As we do not loose any information about the search space when decomposing Σ, the terms *covariance repairing* and *covariance tuning* are used interchangeably with *eigenvalue scaling* and *eigenvalue tuning.*

repair the eigenvalues. Due to the finite precision of decimal numbers represented by computers, it is likely to happen that the eigenvalues will be slightly negative, when the variance among the population shrinks.

Algorithm 5 describes the *Efficient Covariance Matrix Repairing (ECMR)* approach. The basic concept is to search for the smallest eigenvalue λ_{min}. The matrix I is a $D \times D$ identity matrix. After the multiplication $\mid \lambda_{min} \mid \cdot I$, the diagonal of I, will have the value of the smallest eigenvalue. By adding I to V, we can ensure that all former negative eigenvalues will have at least a value of 0 after the addition.

The ECMR0 approach is described in Algorithm 6. This variant searches along the diagonal of V and replaces all negative eigenvalues with 0.

Covariance tuning

Here it is not the covariance matrix itself that is tuned, but rather the eigenvalues, which represent the search space captured by Σ. First of all, covariance tuning can have a positive impact on the exploration nature of the population. The eigenvalues of Σ represent its variance. Thus, by covariance tuning, the variance of the newly sampled individuals can be increased. This trickles down to the problem of convergence to sub-optimal regions, as described above. Secondly, covariance scaling solves a problem caused by the covariance repairing approach described above: ECMR will set at least one eigenvalue to 0, and ECMR0 is likely to set several eigenvalues to zero. When sampling new individuals, the sampled values for dimensions with a zero eigenvalue would be equal to the mean of the respective dimension, since the variance is zero.

2.8.3 Decomposition methods in Evolutionary Computation

It is well known that EDAs perform well on lower dimensional problems, but lose their power quickly when their dimensionality increases. Thus, the performance of EDAs should improve when you disassemble a high dimensional problem into several lower dimensional ones. However, this is not in any case easily possible. Whereas the global optimum of a fully separable function can be found when optimizing each dimension on its own, this may not be possible for problems with epistatic links between decision variables.

In this section we will describe three methods that decompose a large scale optimization problem into several independent sub-problems. Not all of the proposed methods have employed EDA as optimizer. The problem decomposition approach is widely considered as a form of cooperative coevolution as the population consists of several independent sub-populations.

Description of the algorithms

Even though many co-evolutionary algorithms for continuous optimization have been proposed, we focus on three algorithms here, which are more relevant to our work than the other and refer the readers to the literature for the others. To represent several levels of interaction learning, we have decided to analyse the following methods: (i) A random grouping approach, which is from a computational perspective the most efficient one, but does not have the possibility of grouping variables according to epistatic links. (ii) The second approach analyses the correlation among decision variables. This method is slightly more deterministic than the first one, but does not spend any function evaluations on the learning process. Thus, the full amount of function evaluations (FES) can be devoted to the optimization stage. (iii) The third approach deterministically learns the interactions among variables. This method is expected to deliver close to exact results, but requires a large amount of function evaluations for the learning process.

Multilevel Cooperate Co-evolution (MLCC)

MLCC, which has been proposed in [88] is a framework that groups the decision variables of a problem randomly, the pseudocode of this algorithm is illustrated in Algorithm 7. It defines a set of group sizes denoted as G_s and a performance list denoted by P_l. For each group size $c \in G_s$ a performance record $r_k \in P_l$ is connected and group size is randomly chosen in each generation. All decision variables are assigned to m independent groups, where m is determined by the randomly selected group size c so that $D = c \cdot m$. After all independent groups have been optimized separately, the probability r_k determining the chance that group size c is selected in further generation again, is updated according to Equation 2.18. v denotes the best fitness of the last generation and v' the best fitness of the current generation. Assuming a minimization problem is being solved here and elitism in use, equation 2.18 can never be negative. Also v should not be equal to zero. Thus, the algorithm will favour the group size resulting in the best improvement rate. Step (7) of algorithm 7 is optimising the sub-populations of the m independent groups which where grouped in such a way that $D = c * m$, where c is the size of each sub-group m.

$$r_k = \frac{v - v'}{\mid v \mid} \tag{2.18}$$

EDA with Model Complexity Control (EDA-MCC)

The EDA-MCC approach, as proposed in [22], employs a more deterministic algorithm to group variables than MLCC described in the previous sub-section. Algorithms 8-9a-9b summarize EDA-MCC. Firstly it splits all decision variables into two independent subsets: Set W, containing weakly dependent variables (decision variables with only minor interaction with other variables) and set Z containing the strongly dependent variables. For a variable to be either weakly or strongly dependent is determined by the following mechanism: If the absolute value of the correlation of variable X_i with every other variable X_j where $j = 1...D$ and $i \neq j$ (because the correlation of variable with itself is always 1) is below a certain threshold θ, X_i is considered as being weakly dependent. But, if the correlation of X_i with at least one other variable exceeds θ, it is considered as being

Algorithm 7 The Pseudocode of MLCC

```
(1) Set $G_s \leftarrow \{c_1, ..., c_t\}$ *Each* $c_i \in G_s$ determines a specific group size.
(2) Set $P_l \leftarrow \{r_1, ..., r_t\}$ *Each* $r_i \in P_l$ is a performance measure for each group size $c_i \in G_s$.
(3) Set $Pr \leftarrow \{pr_1, ..., pr_t\}$ Each $pr_i \in Pr$ is the probability that group size $c_i \in G_s$ is selected during an optimization cycle. Initially, each value of $Pr$ is set to 1.
(4) $Pr \leftarrow$ Determine the initial selection probability as described in [88].
**Do**
        (5) $c \leftarrow$ select group size for this generation based on $Pr$
        (6) $P_{grouped} \leftarrow \{P_1, ..., P_m\}$ Randomly group the decision variables population
            $P$ into $m$ independent group so that $D = c \cdot m$
        (7) **for each** $P_i \in P$ **do**
            $P_i^* \leftarrow$ Optimise sub-population $P_i$ of the $m$ independent sub-groups
            **endforeach**
        (8) $P_l \leftarrow$ Update the entry $r_k$ in $P_l$ corresponding to group size $c$ according to
            Equation 2.18
        (9) $g \leftarrow g + 1$
**Until stopping criteria are met**
```

strongly dependent.

As all $X_i \in W$ are supposed to be only weakly interacting with other decision variables, they are optimized using a univariate approach. All $X_i \in Z$ are supposed to be interacting with other decision variables. Thus, a multivariate approach is applied to optimize the variables belonging to this subset. As the size of Z is likely to be too large to be optimised successfully from a probabilistic model $\mathcal{M}$, Z is further split into m disjoint sets of decision variables $\{Z_1, ..., Z_m\}$ based on a predefined group size c, such that $| Z |= c \cdot m$.

Cooperative Co-evolution with Variable Interaction Learning (CCVIL)

In [83], *Weicker et al* have proposed a deterministic method to uncover variable interactions between decision variables, which has later been extended to the CCVIL framework by *Chen et al.* in [80]. In contrast to the other algorithms described in this section, CCVIL tries to explicitly find variable interactions. To do so, the authors employ a simple mathematical technique in the decomposition step to discover interactions between variables. This technique is summarized as follows:

Algorithm 8 EDA-MCC

```
Inputs: $\theta$, $c$, $mc$, $sampling$
(1) Set $g \leftarrow 0$.
(2) Set $P \leftarrow$ Generate $N$ points uniformly randomly in the search box to give an initial population.
Do
    (3) Evaluate the fitness of all $N$ points in $P$
    (4) $P^{sel} \leftarrow$ Select the fittest $m < N$ individuals from $P$ using truncation selection.
    (5) Split the search variables in 2 groups:
        (a) Estimate the $D \times D$ correlation matrix $C$ from a random subset of size $mc \leqslant m$ of $P^{sel}$
        (b) Split $\{1, ..., d\} = T_u \bigcup T_s$ as follows:
                $T_u \leftarrow \{i : \forall j \neq i, C(i,j) < \theta\}$
                $T_z \leftarrow \{1, ..., d\} - T_u$
    (6) $W_u \leftarrow P^{sel}_{|T_u}$    $//P^{sel}$ restricted to variables in $T_u$
        $W_z \leftarrow P^{sel}_{|T_z}$    $//P^{sel}$ restricted to variables in $T_z$
        $P^{new}_{|T_z} \leftarrow$ call SM$(W_z, c, sampling)$
        $P^{new}_{|T_u} \leftarrow$ call WI$(W_u)$
    (7) $P \leftarrow P^{new}$
Until Termination criteria are met
Output: $P$
```

Algorithm 9a Subspace Modeling of strongly correlated variables

```
function SM
Inputs: $W_z$, $c$, $smp$
    $L \leftarrow$ dimensionality of $W_z$
    Randomly partition the $L$ variables of $W_z$ into $L/c$
      non-intersecting subsets, $W_{z_1}$,...,$W_{z_{L/c}}$
    for i = 1 to $L/c$
        (a) $\mu_i \leftarrow$ sample mean from $W_{z_i}$
        (b) $\mathbf{\Sigma}_i \leftarrow$ sample covariance $(c \times c)$ from $W_{z_i}$
        (c) If $smp$ = 'Gaussian', $\mathbf{z}_1^{(i)}, ..., \mathbf{z}_N^{(i)} \overset{iid}{\sim} N(\mu_i, \mathbf{\Sigma}_i)$
            Else $smp$ = 'Cauchy' with $\mu_i$ as location
              parameter & $\mathbf{\Sigma}_i$ as dispersion parameter:
              $\mathbf{z}_1^{(i)}, ..., \mathbf{z}_N^{(i)} \overset{iid}{\sim}$ Cauchy$(\mu_i, \mathbf{\Sigma}_i)$
        (d) $Z_{|(i-1)\cdot c+1:i\cdot c} \leftarrow [\mathbf{z}_1^{(i)}, ..., \mathbf{z}_N^{(i)}]$
    endfor
Output: $Z$
end function
```

Algorithm 9b Univariate Modeling of weakly correlated variables

```
function WI
Inputs: $W_u$
    $L \leftarrow$ dimensionality of $W_u$
    for i = 1 to $L$
      (a) Estimate $\mu_i \leftarrow$ sample_mean($W_{u|i}$)
      (b) Estimate $\sigma_i^2 =$ sample_variance($W_{u|i}$)
      (c) Draw $u_1^i, ..., u_N^i \overset{iid}{\sim} N(\mu_i, \sigma_i^2)$
      (d) $U_{|i} \leftarrow (u_1^i, ..., u_N^i)$
    endfor
Output: $U$
end function
```

Assuming x_{best}^k is the vector containing the best values for each decision variable found so far. After optimizing dimension i coevolutionarily, the best individual x_{new}^k in the population *pcc* of the optimizer only focusing on dimension i is extracted as well as a random candidate x_{random}^k from the global population P (with $x_{new}^k \neq x_{random}^k \neq x_{best}^k$). Two new individuals, x_a^k and x_b^k are determined according to Equations 2.19 and 2.20 respectively, subject to X_{new}^k, x_{random}^k and x_{best}^k. This is to test whether dimensions i and j interact. See figure for the illustration of the idea and equation 16 in [60].

$$x_a^k = \begin{cases} x_{new}^k & \text{if } k = i \\ x_{best}^k & Otherwise \end{cases} \tag{2.19}$$

$$x_b^k = \begin{cases} x_{new}^k & \text{if } k = i \\ x_{random}^k & \text{if } k = j \\ x_{best}^k & Otherwise \end{cases} \tag{2.20}$$

As can be seen, individual x_a receives the best known values for each decision variable, since the value at index j of x_a is better than the jth value of x_b^k. In contrast to that, individual x_b received the best known values as well, except for its jth decision variable, which inherits the value from a randomly selected individual of lesser quality. If the inequality $f(x_b) < f(x_a)$ holds, then there is likely to be an interaction between the ith and the jth decision variable [83].

In the CCVIL framework, this approach is implemented as follows (the pseudo code of CCVIL can be found in Algorithm 10). Each individual $X_i \in P$ is a d-dimensional vector of decision variables $\{x_1, ..., x_D\}$. Vector G is d-dimensional and G's ith element stores the group to which the ith decision variable of each individual belongs to. At the beginning, all decision variables are considered as being separable, thus G is initialized to $\{1, 2, 3, ..., D\}$. In each generation of CCVIL's learning phase, a random permutation of x, from the range of $\{1, 2, 3, ..., D\}$ is created, and each element of x refers to a decision variable. For each two sequenced decision variables in x the test for variable interaction as described above is made. If an interaction between both variables is likely, their respective groups in G are merged.

Algorithm 10 The Pseudocode of CCVIL

```
$P \leftarrow$ Initialize random population uniformly across search space.

$G \leftarrow \{1, 2, 3, ..., D\}$ .

% Learning stage
**While** stopping criteria not met **do**
    $Rd \leftarrow$ random permutation $\{1, 2, 3, ..., D\}$
    **for each** element $x_i \in Rd$ **do**
        $j \leftarrow i + 1$
            **if** dimensions denoted by $Rd^i$ and $Rd^j$ are interacting **do**
                $G \leftarrow$ set G's $Rd^{i\prime}s$ and $Rd^{j\prime}s$ value to the same number (merge groups of
                both variables)
            **end**
        **end**
**end**

% Optimization stage
**While** stopping criteria not met **do**
    **for each** distinct group $G_i \in G$ **do**
        $G^* \leftarrow$ Optimise all decision variables belonging to group $G_i$
    **end**
**end**
```

There are other decompositions methods such as Differential grouping [60], which is a variable grouping algorithm that detects interacting variables in black-box optimization problems. It goes by identifying the non-separable groups, which can then be utilised by a cooperative co-evolutionary framework to form the subcomponents. See the theorem in page 423, equations 1 and 2 of [60].

2.8.4 Covariance Matrix Adaption - Evolutionary Strategy (CMA-ES)

CMA-ES is widely considered as one of the leading optimization techniques [13]. Even though it is titled as an Evolutionary Strategy, its main concepts are very similar to the ones of a regular EDA [48]. CMA-ES and its variants have been very successfully used on many low-dimensional problems (e.g.[30], [72]), but benchmarks on high dimensional problems are rather rare. In this section, we will outline the important aspects of the original CMA-ES algorithm and a variant of it called sep-CMA-ES, which has a lower time complexity.

CMA-ES- An outline

Here we discuss the main points of this algorithm. This approach, whose Pseudocode is shown in Algorithm 11, employs manifold features, which partially require an extensive amount of explanation. For this reason, we will focus on some important aspects and refer the interested reader to [35] where Hansen discusses all important aspects in great detail.

In contrast to most EDA implementations, CMA-ES does not initialize the population uniformly across the search space, but rather initialises the starting individuals around a fixed point. Thus, the initial population does not cover a large part of the search space. This requires the population to traverse a lot through the space and, thus, a larger number of generations should be required to explore all regions carefully. CMA-ES employs a small population size, N, which can be calculated by a rule of thumb according to Equation 2.21, where D refers to the dimensionality of the problem.

$$N = 4 + \lfloor 3 \cdot \log(D) \rfloor \tag{2.21}$$

Sampling new individuals is done by adding small normally distributed random numbers to the mean of the selected population, P^{sel}, based on Equation 2.22. Σ refers to the covariance matrix of the selected current population and σ is the step size (standard deviation).

$$P^{new} = mean(P^{sel}) + \sigma \cdot \mathcal{N}(0, \Sigma) \tag{2.22}$$

Naturally, the estimation of a covariance matrix for a high dimensional problem becomes unreliable when only a small number of individuals are available. To increase the reliability of the estimation, the covariance matrices from previous generations are taken into account as well. The adaptation strategy is outlined in Equation 2.23. Σ_{g+1} refers to the covariance matrix used for sampling in the next generation. c_{cov} is a learning rate between 0 and 1. Σ_g is the covariance matrix of the current generation, where in the first generation $\Sigma_0 = I \cdot \Sigma_{g+1}^{\tilde{N}}$ refers to the covariance matrix estimated from $\tilde{N}$ selected individuals of the current generation. A reliable, but simpler estimator for Σ would be the mean of all previous covariance matrices. But, the process described by Equation 2.23 has the advantage, that the old covariances are slowly fading out and, thus, the algorithm has a better learning ability.

$$\Sigma_{g+1} = (1 - c_{cov}) \cdot \Sigma_g + c_{cov} \cdot \frac{1}{\sigma_g^2} \cdot \Sigma_{g+1}^{\tilde{N}} \tag{2.23}$$

The covariance matrix estimated by Equation 2.23 would lead to a very fast convergence to suboptimal regions. Due to the small population size and the limited region covered by the population, it can naturally only represent a small part of the search space. As the sampling is still largely based on the best individuals found so far, a sub-optimal initialization is likely to cause the algorithm to get stuck in this region. Thus, the concept of an evolution path has been introduced. The evolution path is represented by the mean shift of the population of the last generations.

$$P_{g+1}^{c} = (1 - c_c) \cdot P_g^c + \sqrt{c_c \cdot (2 - c_c) \cdot \mu_{eff}} \cdot \frac{mean(P_{g+1}) - mean(P_g)}{\sigma_g} \tag{2.24}$$

The evolution path is calculated according to Equation 2.24. c_c is a learning rate. μ_{eff} is a scaling factor. P_g^c and P_{g+1}^c are the evolution paths at generations g and $g+1$ respectively. $mean(P_g)$ and $mean(P_{g+1})$ are the distribution means of the current generation, g and the next, $g+1$. So the difference, $mean(P_{g+1}) - mean(P_g)$ is the displacement of the distribution means. The term $\sqrt{c_c \cdot (2 - c_c) \cdot \mu_{eff}}$ is used for normalization and $(1 - c_c)$ is the decay factor [35]. By taking the mean shifts into account, CMA-ES can explore the path the best solutions found during the last generations are going. This procedure is combined with the estimation of the covariance matrix according to Equation 2.25. c_{cov} is another scaling factor, which is normally set to the value of μ_{eff}.

$$\Sigma_{g+1} = (1 - c_{cov}) \cdot \Sigma_g + \frac{c_{cov}}{\mu_{cov}} \cdot (P_{g+1}^c) \cdot (P_{g+1}^c)^T + c_{cov} \cdot \left(1 - \frac{1}{\mu_{cov}}\right) \cdot \Sigma_{g+1}^{\tilde{N}} \tag{2.25}$$

By this calculation of the covariance matrix, CMA-ES is able to adjust the shape of the covariance matrix to the landscape of the fitness function. By doing so, it is able to analyse gradient information and, thus, can direct the search to the most promising areas [13], [48]. A graphical illustration of this procedure can be found in [40].

Algorithm 11 The Pseudocode of CMA-ES

```
$D \leftarrow$ Dimension of the problem.
$N \leftarrow$ Offspring population size $(4.0 + 3.0\log(D))$ .
$P^{sel} \leftarrow$ Parent population for next generation $(floor(N/2))$
$\sigma \leftarrow$ Initial standard deviation.
$c_{cov} \leftarrow$ Covariance learning rate.
While stopping criteria not met do
    Update the Covariance Matrix $\Sigma_{g+1}$ according to 2.25
    Update the step size $\sigma_{g+1}$
    Generate Sample Population for generation $g + 1$ according to 2.22
    Update the mean for generation $g + 1$
    Update best ever solution
end While
```

sep-CMA-ES

The sampling step described in Equation 2.22 requires eigendecomposition, and this procedure requires a considerable amount of computational time. In [71], a CMA-ES variant was proposed, which does not sample from a full covariance matrix, but rather from a diagonal matrix. This procedure replaces the requirement to perform eigendecomposition for sampling. A detailed discussion of the computational complexity can be found in [71] as well.

The diagonal covariance matrix only represents the variance among the population, which gets scaled by the evolution path in each generation. Thus, by only considering the variance, sep-CMA-ES looses the capability to explore dependencies.

2.8.5 Random Projection Ensemble EDA (RP-Ens-EDA)

Another divide and conquer technique is the recently proposed Random Projections Ensemble EDA (RP-Ens-EDA). This is an approach, which is state of the art and was proposed in [47]. It introduces an ensemble of random projections (RP) to low dimensions. This way the full covariance is compressed but no correlations are explicitly discarded. The compression is done on the set of fittest search points. The estimation and sampling job is done in the low dimensional space instead of high dimensional space, which makes it efficient. The new population is created and returned to live in the full search space by combining populations from several low dimensional subspaces [47]. The Pseudocode of RP-ENS-EDA is shown in Algorithm 12 below.

Algorithm 12 Pseudocode of the Multivariate Gaussian Random Projection Ensemble EDA (RP-ENS-EDA)

```
(1) Set $g \leftarrow 0$.
(2) Set $P \leftarrow$ Generate $N$ points randomly to give an initial population.
Do
    (3) Evaluate fitness for all $N$ points in $P$
    (4) Select some individuals $P^{sel}$ from $P$
    (5) Estimate $\mu := mean(P^{sel})$
    (6) Generate $\tilde{M}$ independent random matrices $R_i$ with entries drawn i.i.d from Gaussian with mean 0 and variance $\frac{1}{d}$
    (7) For $i = 1, ..., \tilde{M}$
        (a) Project the centred points into k-dimensions:
            $\mathbf{Y}^{R_i} := [R_i(x_n - \mu); n = 1, ..., \tilde{N}]$.
        (b) Estimate the $k$ x $k$ sample covariance $\Sigma^{R_i}$.
        (c) Sample $N$ new points $y_1^{R_i}, ..., y_N^{R_i} \sim_{i.i.d} N(0, \Sigma^{R_i})$.
    EndFor
    (8) Let the new population $P^{new} := \sqrt{\frac{d\tilde{M}}{k}}[\frac{1}{\tilde{M}}\Sigma_{i=1}^{\tilde{M}} R_i^T y_1^{R_i}, ..., \frac{1}{\tilde{M}}\Sigma_{i=1}^{\tilde{M}} R_i^T y_N^{R_i}] + \mu$.

    (9) $P \leftarrow P^{new}$
    (10) $g \leftarrow g + 1$
Until Termination criteria are met

    (11) Output $P$
```

RP-Ens-EDA proceeds by initially generating a population of individuals randomly everywhere in the search box and selects the $\tilde{N}$ fittest points based on their fitness values. This is represented as P^{sel} in algorithm 12. The number of subspaces is denoted by $\tilde{M}$, which is a parameter. These subspaces are created in order to project the fittest individuals down to these subspaces with dimensionality $k \ll D$. The authors in [47] choose to take $\tilde{M} = 4D/k$ where D is the dimension of the original space and $k = 3$ is the dimension of the subspaces. The latter is also a parameter of the method. They also have other input parameters such as the population size N and the maximum fitness evaluation allowed MaxFE. Once the P^{sel} is determined, its mean is estimated in step (5) to be used in centering the points. Since they are going to have $\tilde{M}$ subspaces, $\tilde{M}$ independent random projection matrices are generated in step (6) so as to project the fittest individuals down to k dimensions in these $\tilde{M}$ subspaces. The default option of RP matrices is to use i.i.d. Gaussian entries. Step 7(a) projects the fittest individuals down to the subspaces of dimension k, then estimates the $k \times k$ covariance matrices for each of the subspaces and samples N new points in each subspace using the multivariate Gaussian search distribution. Step (8) averages the individuals obtained from the different subspaces to produce new population P. This is the 'combine' stage of the algorithm, which outputs the new population, and it can be shown that, conditioned on the projection matrices, this new population is distributed as a multivariate Gaussian with covariance that is a regularised version of the sample covariance. This process is repeated until a stopping criterion is met.

2.9 Inadequacies of existing methods

We just looked at a few of the most relevant methods to our work. In this section, we will highlight some of their advantages and disadvantages.

Current practices of EDA are geared towards improving the exploration abilities of the algorithms by reducing the complexity of conducting the probabilistic model and considering independent variables as it is in the univariate EDA (UMDAc) in [81]. Intuitively, univariate EDA will perform well on separable problems, but due to the simplistic way of building its model, we expect a significant performance deterioration on problems with high amount of interactions if univariate EDA is applied to non-separable problems. It is known that although the exploration ability for a univariate model will be improved slightly, it cannot adequately tackle non-separable problems, which is a short fall for this method. The authors in [50] and [57] proved this fact theoretically and empirical confirmation of the fact is featured in [28].

Literature has shown that the Gaussian distribution as a search operator is prone to premature convergence when the population is far from the optimum [73]. Recent work, [86] suggests that replacing the univariate Gaussian with a univariate Cauchy distribution in EDA holds promise in alleviating this problem because it is able to make larger jumps in the search space due to Cauchy distribution's heavy tails. However, methods employing this suggestion will not take into consideration the interactions between decision variables as mentioned above. We see this as an inadequacy in the usage of Cauchy distribution as an alternative search distribution to Gaussian. Therefore, extending this to a multivariate setting to combine the advantages of multivariate modelling with the ability of escaping early convergence can lead methods to efficiently explore the search space.

Authors of [46] applied ensemble of random projections technique on the set of fittest search points to low dimensions and did estimation and sampling in the low dimensional space instead of doing them in the high dimensional space, which makes it efficient. We have explored this technique further and found some rooms to improve on. The authors generate the entries of the random projection matrices from Gaussian, but generating these entries from a heavy tail distribution instead will control the size of the covariance in order to balance exploration and exploitation of the search process, which won't be possible when the entries of the random projection matrices are from Gaussian distribution.

Model building in high dimensions is the subject of many recent research efforts, as high dimensionality limits the usefulness of optimisers in practice. Many approaches were proposed ranging from decomposition method [24] to methods that use limited dependencies [83]. All the methods that use the above techniques used the whole dimensions in their optimisation process to scale the methods up. However, it has been noted that in certain classes of functions most decision variables have a limited impact on the objective function [82]. These type of functions are said to have intrinsic dimensions. It is possible to exploit these intrinsic dimensions of these functions without knowing the influential subspaces of the input spaces, or their dimensions and optimise the functions considering only the intrinsic dimensions to mitigate the curse of dimensionality. The existing methods did not take this into consideration. We can achieve this by employing the idea of random embedding.

CHAPTER 3

Multivariate Cauchy EDA Optimisation

In this chapter[1], we developed a new EDA variant which samples from a multivariate Cauchy instead of sampling from the commonly used multivariate Gaussian which is prone to premature convergence. A research work suggests that replacing the univariate Gaussian with a univariate Cauchy distribution in EDA holds promise in alleviating this problem because it is able to make larger jumps in the search space due to the Cauchy distribution heavy tails. It is in this light that we proposed to use multivariate Cauchy distribution in this research to blend together the advantages of multivariate modelling with the ability of escaping early convergence to efficiently explore the search space. We conducted extensive experiments on 2D problems to establish whether Cauchy is better than Gaussian. We did some comparison between the two search distributions on 16 benchmark functions and we found that multivariate Cauchy EDA is better than univariate Cauchy EDA, and we have also found multivariate Cauchy being advantageous over multivariate Gaussian EDA when the population lies far from the optimum.

[1]A shorter version of this work presented in this chapter appears in Proc. IEEE International conference on Intelligent Data Engineering and Automated Learning (IDEAL) 2014, LNCS 8669, pp. 449-456, (c) Springer, 2014.

3.1 Introduction

In classical EDA, Gaussian distribution is used as the search operator to build a probabilistic model to fit the fittest individuals and create new individuals by sampling from the created model. It has been established that Gaussian EDA is prone to premature convergence [47], [49], [69] when its parameters are estimated using the maximum likelihood estimation (MLE) method. It converges too fast and does not get to the global optimum. Figures 3.2 and 3.3 on pages 61 and 62 respectively illustrate this fact. Figures 3.2 and 3.3 are movie plots showing the instances of the evolutionary process of Cauchy and Gaussian at each generation. In the experiments of these figures, the populations were initialised far from the optimum. This is to test the advantage of Cauchy's heavy tail, which enables it to make long jumps, thus escaping premature convergence. It is also meant to test the premature convergence associated to Gaussian. From the plots, you can see that Cauchy has already found the global optimum at its 11th generation, while Gaussian got stuck at the 10th or 11th generation and has never reach the global optimum.

The premature convergence of classical Gaussian EDA attracted many efforts geared towards solving this problem to avail EDA the chance of escaping it [69],[87]. Here we present the usage of multivariate Cauchy distribution, an extension of [86] with a full matrix valued parameter that encodes dependencies between the search variable as an alternative search operator in EDA. We utilize its capability of making long jumps so as to escape premature convergence, which is typical of Gaussian in order to enable EDA algorithms get to the global optimum. Although Cauchy has already been used as an alternative search distribution for EDA [65], [69] and [86], it was the univariate version of Cauchy that was utilized, discarding statistical dependences among the search variables. We compare the performance of multivariate Gaussian EDA with multivariate

Cauchy EDA to establish whether, the long jumps that Cauchy is able to make will be advantageous. We also compare the performance to Univariate Cauchy EDA to establish advantage of multivariate modelling.

3.2 Algorithm Presentation

The algorithms used in this chapter for comparison are multivariate Gaussian EDA (MGEDA), multivariate Cauchy EDA (MCEDA) and Univariate Cauchy EDA (UCEDA). MGEDA takes on board correlation between variables in the selected individuals through its full covariance matrix, and MCEDA encodes pairwise dependencies among the search variables through its matrix valued parameter. UCEDA neglects dependences among the search variables. The classical EDA algorithm is shown in algorithm 2 of chapter 2.

Algorithm 2 is a typical EDA, which proceeds by initially generating a population of individuals and then evaluates their fitness to select the fittest ones based on their fitness using the truncation selection. For the MGEDA, we compute the maximum likelihood estimates (MLE) of the mean (μ) and the covariance (Σ) of the fittest individuals and use these parameters to generate new ones by sampling from a multivariate Gaussian distribution with parameters μ and Σ. For MCEDA, we use the same estimates to sample from a multivariate Cauchy distribution in step 6. In UCEDA, we use μ and the diagonal elements of Σ to sample each from Univariate Cauchy. The new populations from the three algorithms are formed by replacing the old individuals by the new ones. In MCEDA, we make use of the Gaussian scale-mixture representation of the Cauchy density [63] to sample from Cauchy. The following equation gives the expression of multivariate Cauchy:

$$\text{Cauchy}_x(\mu, \Sigma) = \int_{u>0} N_x\left(\mu, \Sigma/u\right) \text{Ga}_u\left(1/2, 1/2\right) du \tag{3.1}$$

where u may be regarded as an hidden variable, and $\text{Ga}(\cdot)$ is the Gamma density. Cauchy distribution is a heavy tailed distribution that is liable to produce large values. They have heavier tails than the exponential distribution. The heavier the tail, the larger the probability that you'll get one or more very large values in a sample.

3.2.1 A note on parameter estimation

The philosophy in EDA is to estimate the density of the selected individuals so that when new individuals are sampled from the model, they will follow the same distribution as the estimated density. Fortunately for Gaussian, this works. Parameter estimation in Cauchy distributions was studied in statistics [63] where an Expectation and Maximization (EM) algorithm was developed to find the exact maximum likelihood estimate of a multivariate Cauchy distribution from a set of points, which we implemented for our study. Multivariate Cauchy density function is defined as follows:

$$f(y;\mu,\Sigma,D) = \frac{\Gamma(\frac{1+D}{2})}{\Gamma(\frac{1}{2})\pi^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}[1+(y-\mu)^T\Sigma^{-1}(y-\mu)]^{\frac{1+D}{2}}} \tag{3.2}$$

Where y is the observed variable, D the dimension of the function, μ the location parameter and Σ a matrix value parameter that encodes pairwise dependencies. Equation 3.2 is the expression of multivariate Cauchy, while equation 3.1 is the scale mixture representation of equation 3.2.

The Pseudocode used to estimates the exact maximum likelihood estimation(MLE) of parameters μ and Σ for Cauchy is shown in Algorithm 13 below, where ν is the degree of freedom. The derivation of μ and Σ for Cauchy is shown in the appendix B.

However, we found that when we estimate the Cauchy parameter (Using EM), then the obtained model of the selected individuals (Cauchy density) will disregard any outliers. This is of course what a robust density estimator is meant to do- However for optimization those outliers maybe some rare and very good solutions that got close to an optimum. Fig 3.1 illustrates such an example. As you can see in fig. 3.1 which was a snap shot taken

Algorithm 13 The Pseudocode for estimating μ and Σ for Cauchy

```
Set $g \leftarrow 0$. Generate $N$ samples.
Estimate $\mu$ and $\Sigma$ .
Do MLE using Expectation-Maximisation(EM) Algorithm.
Repeat N times

    • (E-Step), for each i, Set

      $E(u_i) := \frac{\nu+D}{\nu+\frac{1}{2}(y_i-\hat{\mu})^T \hat{\Sigma^{-1}}(y_i-\hat{\mu})}$

    • (M-Step) Update the parameters

      $\hat{\mu} := \frac{\Sigma_{i=1}^N y_i E(u_i)}{\Sigma_{i=1}^N E(u_i)}$

      $\hat{\Sigma} := \frac{1}{N}\sum_{i=1}^N (y_i - \hat{\mu})(y_i - \hat{\mu})^T E(u_i)$

Set $g \leftarrow g + 1$
end
```

from an iteration of the experiments we conducted, two selected individuals are close to the optimum and as such are good solutions but they are outliers with respect to the rest of the selected individuals. This is the reason why in algorithm 2, the multivariate Cauchy distribution was used only in the sampling step.

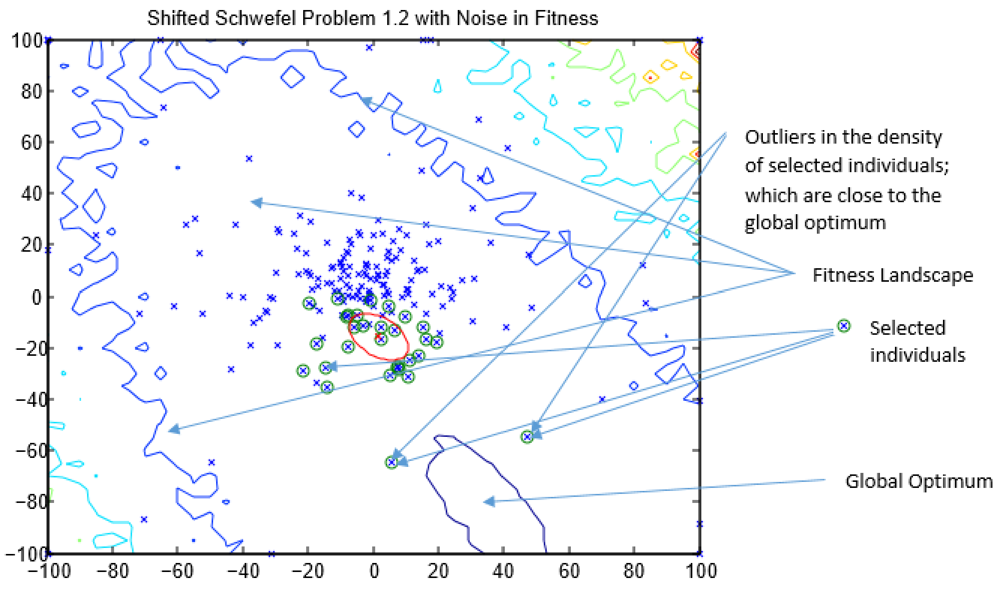


Figure 3.1: A plot showing the behaviour of EDA when the search distribution is a Cauchy distribution.

3.3 Implementation and Experiments

Our hypothesis is that MCEDA has better performance than both UCEDA and MGEDA when the initial population is far from the optimum and also when the population size is small. In turn multivariate Gaussian should perform better when the population is close to the optimum. To test this hypothesis, we conducted an extensive experiment on 16 benchmark functions taken from [77]. In the following subsections, we will describe the functions, parameter settings, then we present results with analysis and conclude.

3.3.1 Benchmark test functions

The comparisons of the three EDA algorithms were carried out on the suite of benchmark functions from the CEC 2005 competition. Details in Appendix A. 16 test functions were used in this experiment. Among the functions tested, 5 are unimodal and 11 multimodal. All the global optima are within the given box constrains. However, problem 7 was without a search range and with the global optimum outside of the specified initialization range. All problems are minimization. Please see details of the functions here [77].

3.3.2 Parameter settings

The dimensionality of all the problems is 2. This is because we want to be able to visualise the optimisation process and its results for understanding and insight. We carried out three sets of experiments. The first experiment was conducted with the initial population size set to 20, which is initialised uniformly randomly everywhere. This is to test the performance of Gaussian and Cauchy on small population size. The second was conducted with a population size of 200 and the third with a population size of 500 respectively. The second experiment has its initial population initialised far from the optimum, while the third experiments has its population initialised uniformly randomly everywhere within the search space. This is to test the performance of Cauchy and Gaussian in these two settings. The percentage of individuals retained is 30% for all the experiments conducted, which is a most widely used selection ratio. We did 25 independent runs for each problem on a fix budget of 10,000 function evaluations in each case. The program terminates after the budget is exhausted for each run for all the 25 runs. The initialization was uniformly random everywhere within the search space for population size of 20 and 500. We also created harder versions of these problems by initializing far from the optimum to establish whether MCEDA can still perform in this situation. The population size used in this experiment was 200. Both Gaussian and Cauchy distributions where applied on EDA. Parameters for all the algorithms are the same. The only difference is in the search distribution.

3.3.3 Performance criteria

The main performance criterion was the difference in fitness values (gap) between the best individual found and the global optimum.

3.4 Results and Discussion

The results of our experiments are summarized in tables 3.1 to 3.3 and bold font indicates statistically significant out performance. Tables 3.1 and 3.2 report results from experiments that compare MCEDA with MGEDA. MCEDA performed better than MGEDA in most of the 16 benchmark functions, see tables 3.1 and 3.2. We also found when the population size was increased to 200, MGEDA outperformed the MCEDA, and this was also confirmed in the results of the experiments with population size 500. The reason for this is the MGEDA is better when the best individuals are close to the optimum, so when we initialized lots of them everywhere, there are chances that some of them will be close. Now considering the cases where MCEDA was outperformed by MGEDA and repeated the experiment with initial population far from the optimum, we can see from the results in table 3.1 that MCEDA has performed better than MGEDA. This is because of the long jumps of Cauchy. In table 3.3, we report comparison between MCEDA and UCEDA. We can clearly see from table 3.3 at 95% confidence interval that MCEDA is better than UCEDA. MCEDA performed better than UCEDA in most of the functions.

Table 3.1: Statistical Comparison of MCEDA and MGEDA on Problems 01-16 with initial Population far from the optimum and has size 200.

	MCEDA		MGEDA		Rank Sum Test	
	Mean	Std	Mean	Std	H	P
P01	**0**	**0**	1.62E+03	508.7709	1	9.72E-11
P02	**0**	**0**	1.41E+03	287.7206	1	9.72E-11
P03	**0**	**0**	3.87E+08	1.24E+08	1	9.72E-11
P04	**0**	**0**	953.7703	183.1595	1	9.72E-11
P05	**0**	**0**	7.09E+03	411.0024	1	9.72E-11
P06	**0**	**0**	8.01E+08	1.59E+08	1	9.72E-11
P07	**0.0621**	**0.0464**	0.7204	0.7672	1	0.001
P08	18.0903	3.9646	19.2004	4.0001	0	0.0625
P09	0.9025	0.6808	1.3412	0.973	0	0.1057
P10	**0.6757**	**0.6134**	3.8126	2.4724	1	4.04E-07
P11	1.1429	0.4346	**0.0359**	**0.0993**	1	2.47E-10
P12	193.2284	184.7226	**0.4156**	**0.6739**	1	3.93E-09
P13	**0.0067**	**0.0147**	23.858	8.4094	1	6.57E-10
P14	**0.022**	**0.0036**	0.7357	0.2264	1	1.41E-09
P15	**0.0649**	**0.0402**	52.371	7.9035	1	1.41E-09
P16	0	0	1.7225	4.6068	1	0.0412

Table 3.2: Statistical Comparison of MCEDA and MGEDA on Problems 01-16 with uniform initialisation and small Population size 20.

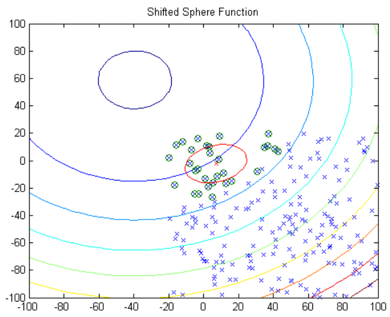
	MCEDA		MGEDA		Rank Sum Test	
	Mean	Std	Mean	Std	H	P
P01	**0**	**0**	35.1160	163.9960	1	4.4787e-009
P02	**0**	**0**	116.3862	213.6704	1	1.3101e-009
P03	**0**	**0**	7.3128e+04	3.5850e+05	1	3.6574e-010
P04	**0**	**0**	132.4800	311.6559	1	1.3101e-009
P05	**0**	**0**	1.4107e+03	1.2712e+03	1	4.4787e-009
P06	**0**	**0**	14.0273	20.0025	1	3.6574e-010
P07	0.0858	0.1346	0.4516	1.1137	0	0.2288
P08	19.7487	4.1438	**17.6353**	**6.5513**	1	2.4248e-008
P09	3.5818	4.0211	0.6134	0.5887	0	0.2111
P10	**0.7562**	**0.9646**	1.3387	1.9365	1	0.0092
P11	0.5856	1.1481	**0.2505**	**0.3295**	1	0.0195
P12	2.0355e+03	1.1826e+03	**10.2325**	**20.2760**	1	9.2160e-010
P13	0.1538	0.2352	0.0507	0.0531	1	0.1633
P14	0.6806	0.2313	**0.1408**	**0.0194**	1	1.4634e-007
P15	0.1183	0.1278	1.1885	2.9844	0	0.8613
P16	**0.6865**	**2.9408**	2.2038	3.8294	1	1.1752e-007

Table 3.3: Statistical Comparison of UCEDA and MCEDA on Problems 01-16 with uniform initialisation and small Population size 20.

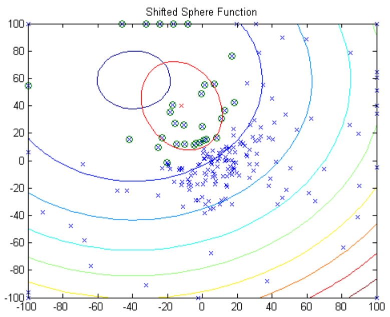
	UCEDA		MCEDA		Rank Sum Test	
	Mean	Std	Mean	Std	H	P
P01	0	0	0	0	0	N/A
P02	0	0	0	0	0	1.N/A
P03	5.2295e+03	6.0844e+03	**0**	**0**	1	9.7282e-011
P04	0	0	0	0	0	N/A
P05	0	0	0	0	0	1.N/A
P06	5.6064	14.1692	**0**	**0**	1	9.7282e-011
P07	0.2708	0.2092	**0.0858**	**0.1346**	0	1.4256e-004
P08	20.5476	0.5120	19.7487	4.1438	0	0.9690
P09	**3.5818**	**4.0211**	6.1954	4.4285	1	0.0289
P10	0.9154	0.9916	0.7562	0.9646	0	0.3855
P11	0.1991	0.5776	0.5856	1.1481	0	0.3731
P12	**1.0293e+03**	**722.7733**	2.0355e+03	1.1826e+03	1	0.0108
P13	0.3313	0.3508	**0.1538**	**0.2352**	1	1.7045e-004
P14	0.8139	0.0550	**0.6806**	**0.2313**	1	0.0085
P15	0.3948	0.1526	**0.1183**	**0.1278**	1	5.0089e-007
P16	0.4866	2.4328	0.6865	2.9408	0	0.5717

3.5 Summary

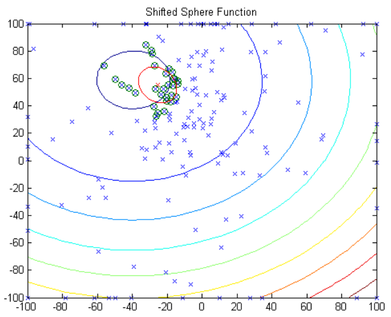
In this chapter, we studied the use of a multivariate Cauchy distribution in black-box continuous optimization by EDA. Our MCEDA blends together the advantages of multivariate modelling with the Cauchy sampling ability of escaping early convergence and efficiently explore the search space. We conducted extensive experiments on 16 benchmark functions and found that MCEDA outperformed MGEDA when the population is far from the global optimum and is able to work even with small population sizes. We also demonstrated the superiority of multivariate Cauchy EDA against univariate Cauchy EDA.



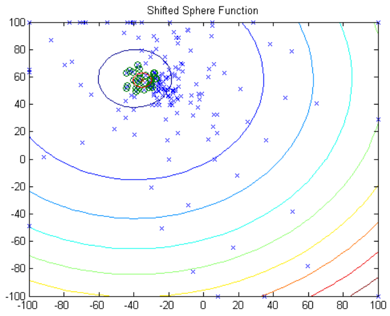
(a) Generation 1



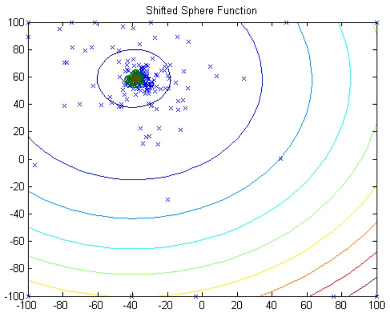
(b) Generation 2



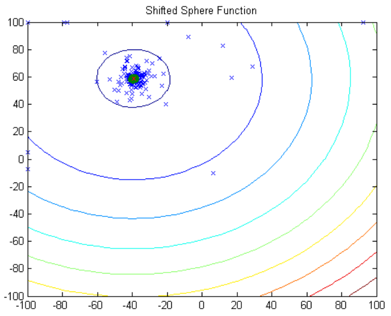
(c) Generation 3



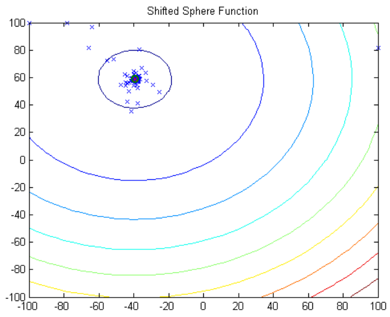
(d) Generation 4



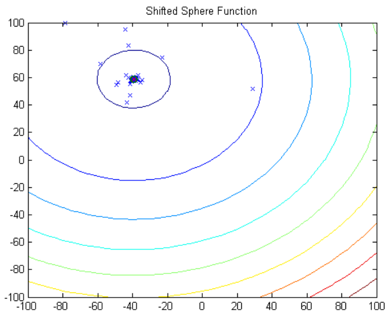
(e) Generation 5



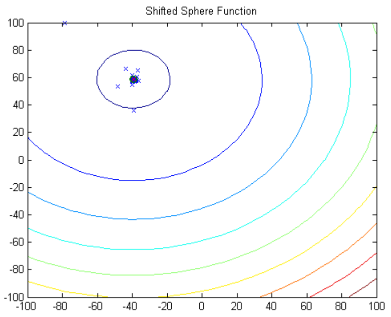
(f) Generation 6



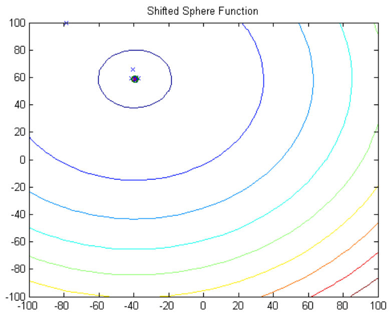
(g) Generation 7



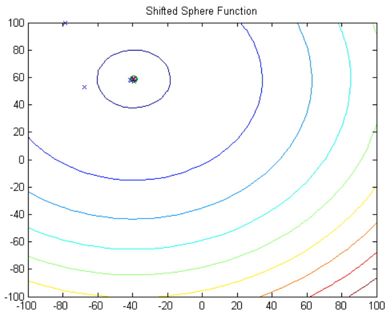
(h) Generation 8



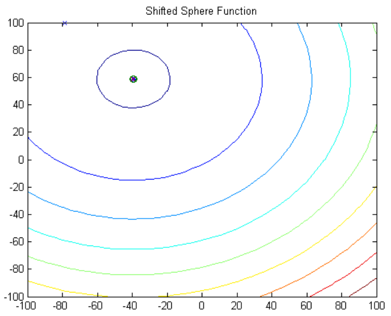
(i) Generation 9



(j) Generation 10

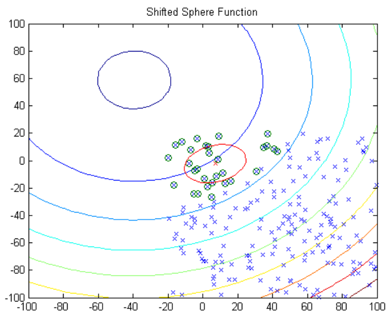


(k) Generation 11

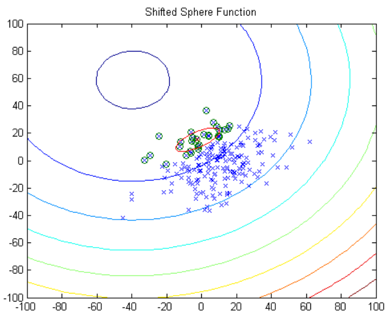


(l) Generation 12

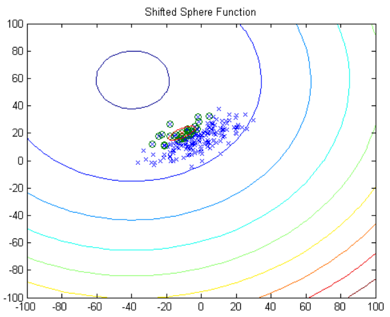
Figure 3.2: The Evolutionary path of Cauchy on 2-dimensional shifted sphere function



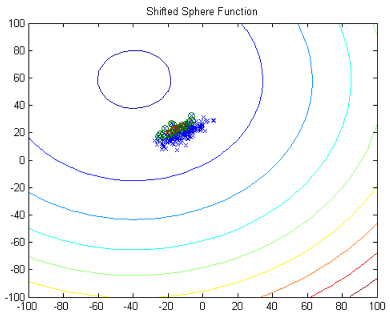
(a) Generation 1



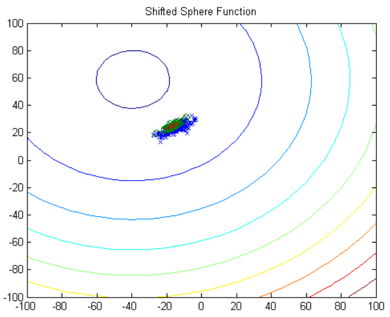
(b) Generation 2



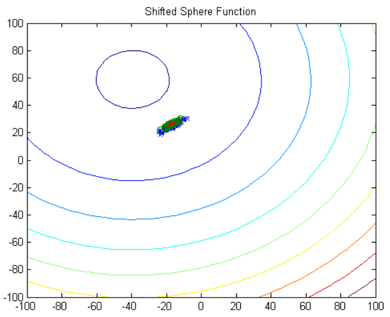
(c) Generation 3



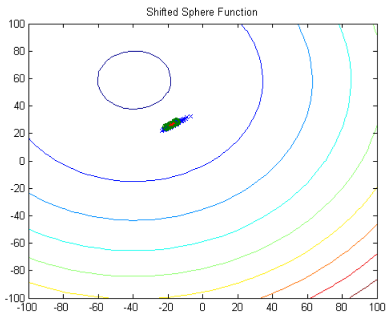
(d) Generation 4



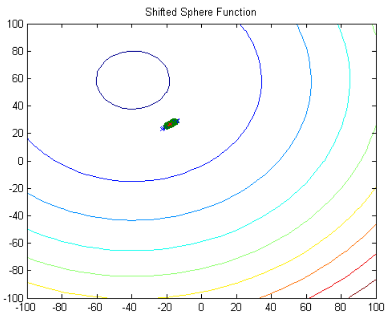
(e) Generation 5



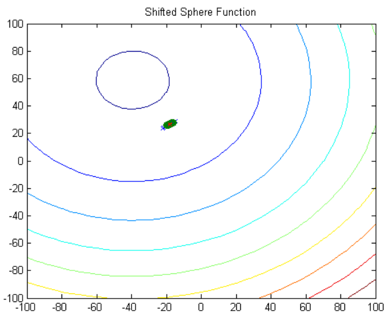
(f) Generation 6



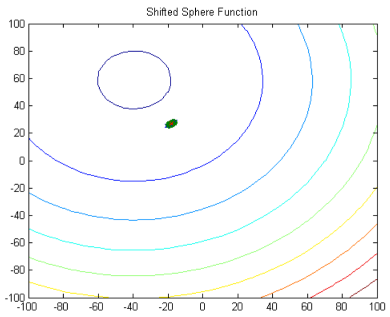
(g) Generation 7



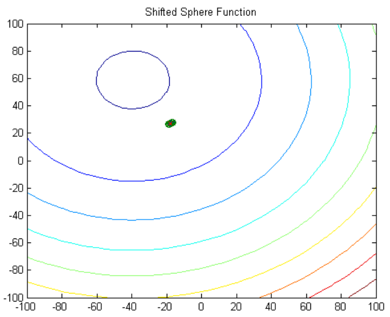
(h) Generation 8



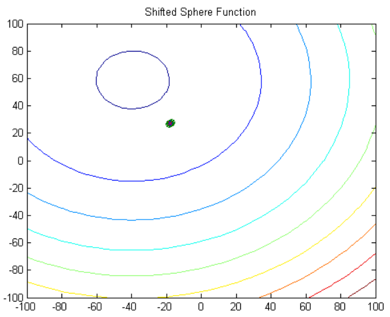
(i) Generation 9



(j) Generation 10



(k) Generation 11



(l) Generation 12

Figure 3.3: The Evolutionary path of Gaussian on 2-dimensional shifted sphere function

CHAPTER 4

How effective is Cauchy-EDA in high dimensions?

This chapter examines the behaviour of the approach taken in chapter 3 on high dimensional problems. Some authors have already suggested that employing a heavy tailed search distribution, such as a Cauchy, may enable EDA to better explore a high dimensional search space [75]. However, other authors have found Cauchy search distributions are less effective than Gaussian search distributions in high dimensional problems [38], [66] and [67]. In this chapter[1], we set out to resolve this controversy. As we shall see, the comparative behaviour of Cauchy vs. Gaussian turns out to be surprising in high dimensions.

4.1 Introduction

EDA is known to have good properties as long as the search space is low dimensional, but it is notoriously bad in high dimensions due to excessive computational resource requirements [22], [47], [58]. In an attempt to remedy this, several authors have proposed employing heavy-tailed distributions in the sampling step of EDA instead of the more

[1]A shorter version of this work presented in this chapter appears in Proc. IEEE International Congress on Evolutionary Computation (CEC) 2016. **This work was nominated for a Best Paper Award.**

commonly used Gaussian. For instance, [81] proposes a univariate continuous EDA (UMDAc) with Lévy sampling. Furthermore, in later work by [75], Cauchy sampling has been reported to be superior to Gaussian in high dimensions. Cauchy is a very heavy tailed distribution that has no finite mean. From the conclusions of these works it appears as though the ability to make long jumps should be beneficial for high dimensional search. Though, we should note that, the study in [75], although termed 'high dimensional' by the authors, it only considered problems of up-to 32 dimensions.

Low dimensional studies (up-to 3 dimensions) are pretty consistent to find Cauchy superior to Gaussian when the population is relatively far from the optimum - see for instance [37], [65], [73], [86] and the previous chapter. But in high dimensions we see a controversy in the existing literature as highlighted in chapters 1 and 2. One issue is that the previous comparisons mentioned in chapters 1 and 2 were done with different algorithms so it is hard to distill a global picture. Secondly, evidence about the merits of Cauchy vs. Gaussian based search is largely missing in the literature on problems larger that 40 dimensions. What will happen on problems with 50-1000 dimensions?

In this chapter we set out to resolve the above controversy, and we conduct a thorough investigation into the performance of multivariate Cauchy EDA in high dimensions up to 1000 dimensional problems in comparison with its Gaussian counterpart. We shall use a scalable variant of EDA called EDA with Model Complexity Control (EDA-MCC) [22] for our purpose, and create a Cauchy sampling variant of it.

4.2 Presentation of the algorithm used in this work

We chose EDA-MCC [22] as the algorithmic tool for our experiments, because it is scalable and applicable to both low and high dimensional problems, and it was previously demonstrated to work well up to 500 dimensions. This allows us to vary the problem size and observe the trends in performance comparatively for Gaussian and Cauchy search distributions.

In its original form, EDA-MCC employs a multivariate Gaussian search distribution, which, for scalability purposes, is modelled/approximated as a product distribution on non-overlapping subspaces. These are created by randomly partitioning the search variables that have correlations into disjoint groups. During the evolutionary process, the algorithm keeps measuring the degree of linear interdependencies between variables by computing their correlation coefficients and the variables that only have correlations smaller than the threshold in absolute value, (meaning the observed linear dependencies between variables are weak) are modelled as univariate product distributions.

Before proceeding further, we should mention that correlation only captures linear dependencies and will miss any non-linear ones. Initially, we also experimented with employing Mutual Information instead [85], but observed no significant improvement, and due to the computationally more demanding scaling we did not pursue it further.

It is straightforward to modify this strategy to sample from independent multivariate Cauchy blocks instead, which we do for the purpose of our experiments. The pseudo-code of a generic EDA is given in Algorithm 2, and the Algorithms 8-9a-9b summarize EDA-MCC which can be found in chapter 2. Our only modification is in the multivariate modeling, namely step (c) of Algorithm 9a, to allow for multivariate Cauchy sampling in the subspaces. We implemented the multivariate Cauchy sampling by making use of the Gaussian scale-mixture representation of the Cauchy density [63] shown earlier in eq. 3.1, and sampling this generatively, in the same way as in chapter 3.

4.3 Experiments

We set out to resolve the controversy about the comparative merits of multivariate Gaussian vs. Cauchy search distributions in high dimensions. Towards this end, we conducted experiments on 7 benchmark functions taken from the CEC'05 competition [77] – these

Table 4.1: Scalable test functions from the CEC'05 collection.

Problem	Name	Type
P01	Shifted Sphere Function	Unimodal
P02	Shifted Schwefel's Problem 1.2	Unimodal
P03	Shifted Rotated High Conditioned Elliptic Function	Unimodal
P04	Shifted Schwefel's Problem 1.2 with Noise in Fitness	Unimodal
P05	Shifted Rosenbrock's Function	Multimodal
P06	Shifted Rastrigin's Function	Multimodal
P07	Expanded Extented Griewank Function plus Rosenbrock	Multimodal

are listed in Table 4.1 – and we varied the problem dimensionality from 20 up to 1000. Among the functions tested, 4 are unimodal, and 3 are multi-modal. All the global optima are within some given box constraints. All problems are minimization. More details on the functions may be found in [77].

4.3.1 Roadmap and parameter settings

Our first experiments were conducted on the Shifted Rosenbrock Function to replicate the findings of [75] in the settings considered there (i.e varying dimensions up to 32). The purpose of these experiments was to see if the version of EDA we are using is consistent with their findings. Once confirmed, we further looked at the Shifted Rosenbrock Function in higher dimensions to get a more complete picture. As we shall see, the conclusion turns out to be very different in the higher dimensional regime.

We then conducted experiments on a good number of benchmark problems to test if the above finding is observed more generally. The following set of dimensions (problem sizes) were used to conduct our experiments, {20, 30, 40, 50, 100, 200, 300, 400, 500, 1000} for all problems.

All experiments were ran with three different population sizes {300, 1000, 2000} in order to make sure that the observed behavior is not a byproduct of a particular choice of population size. A budget of $10000 \times D$ function evaluations was set in all experiments, where D is the dimension of the problem. This was the recommended budget size in [77] for the CEC'05 competition.

The following tunable parameters were set in accordance with the recommendations in [22]: The threshold θ to decide if a search variable has weak or strong correlations is set to 0.3, the number of selected individuals ($\tilde{N}$) is set to half of the population size, and the sample size used to estimate correlations (mc) is set to 100. However, we did not go by the recommendation of [22] in setting the maximum group size, c. The reason will be explained shortly. Instead, we set $c = \min(\lceil D/5 \rceil, \lceil N/15 \rceil)$, where N is the population size. The performance criterion is the difference (gap) between the fitness of the best individual found and the true global optimum. Each experiment was run 25 times (with random independent restarts) and we report the average and standard deviation of these differences.

A note on setting the max group size, c in EDA-MCC

We believe the following must be a typo on page 811 in [22], for their 500-dimensional experiments, where the block size is claimed to be set to $c = 100$ and the number of selected individuals is $\tilde{N} = 100$. In our experience this setting does not work, and indeed this setting would mean to estimate 100×100 covariance blocks from only 100 points which leads to a singular covariance estimate. Thus one needs to either reduce the block size c or to increase the population size N: Since the latter is undesirable we took $c = \min(\lceil D/5 \rceil, \lceil N/15 \rceil)$. With our setting, now we have $c \times c = \min(\lceil D/5 \rceil, \lceil N/15 \rceil) \times \min(\lceil D/5 \rceil, \lceil N/15 \rceil)$ covariance blocks to estimate from $\tilde{N} = \lceil N/2 \rceil$ points. The table of parameters is shown in table 4.2 below with justification of choosing them mention in the text above. We used the default values as in [22] with the exception of the group size, c. We have justified why we choose the value use for c in the section above.

Table 4.2: Table of parameters used in this chapter.

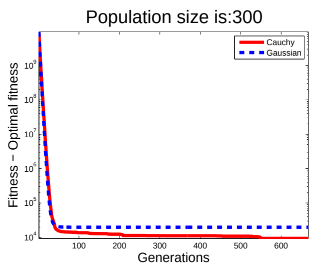
Parameter	Description	Value
D	Problem Dimensionality	{20, 30, 40, 50, 100, 200, 300, 400, 500, 1000}
N	Population Size	{300, 1000, 2000}
θ	Correlation threshold	0.3
$\tilde{N}$	Number of selected individuals	$\lceil N/2 \rceil$
mc	Sample size use to estimate correlations	100
c	Maximum group size	$\min(\lceil D/5 \rceil, \lceil N/15 \rceil) \times \min(\lceil D/5 \rceil, \lceil N/15 \rceil)$
m	Number of non-intersecting subsets	$\lceil \lvert Z \rvert / c \rceil$
$runs$	Number of runs	25
$MaxBudget$	Maximum number of function evaluation	$10000 \times D$

4.4 Results and Discussion

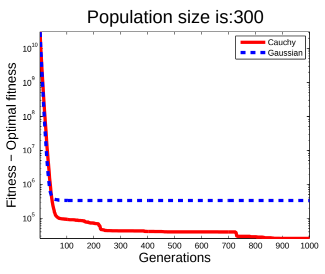
We present results of our experiments in three parts. The first part will consist of results of experiment done on the Rosenbrock function to verify the claim made in [75] and developing more complete picture. The second part contains results of an extensive empirical study on 7 benchmark functions, and the third part examines the case when the optimum is shifted much further away.

4.4.1 Results on shifted Rosenbrock: Confirming the findings of [75], and developing a more complete picture

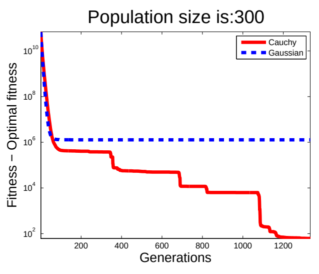
Following [75], we start by running experiments on the shifted Rosenbrock function up to 32 dimensions. As we already mentioned, [75] reported superior performance when employing the Cauchy search distribution as opposed to the Gaussian when tested in this dimensionality range. Although they use a different optimization algorithm and different parameter setting than ours, we were able to confirm their finding. Figures 4.1 - 4.3 show the results obtained with population sizes 300, 1000 and 2000 respectively and Tables 4.3 - 4.5 show the corresponding statistical analysis. We see the Cauchy search distribution performs significantly better than the Gaussian up to 50 dimensions for the case of experiment ran with population size of 300 and up to 100 dimensions in the case of experiment ran with population sizes of 1000 and 2000.



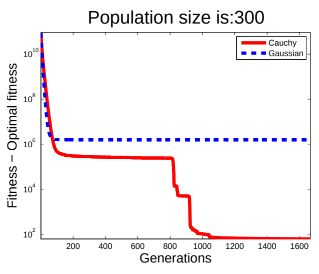
(a) Dimension 20



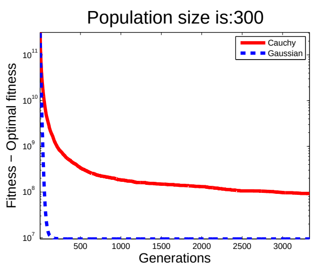
(b) Dimension 30



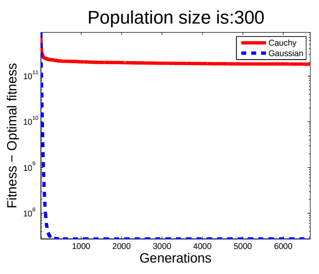
(c) Dimension 40



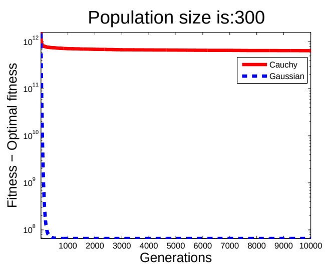
(d) Dimension 50



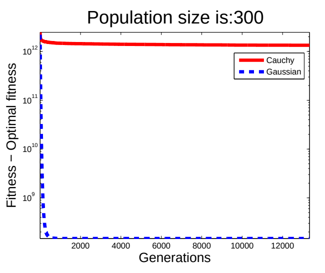
(e) Dimension 100



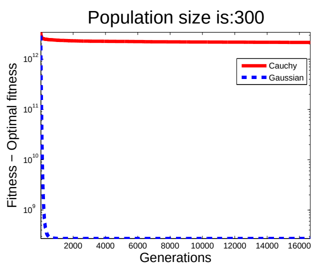
(f) Dimension 200



(g) Dimension 300



(h) Dimension 400



(i) Dimension 500

Figure 4.1: Convergence behaviour of Gaussian and Cauchy as search distribution for EDA-MCC on different dimensions of the Rosenbrock function for population size of 300 on average, as obtained from 25 independent runs. A total budget of $1 \cdot 10^4 \cdot D$, where D is the dimension of the problems were used.

However, we also see from figures 4.1 - 4.3 and Tables 4.3 - 4.5 that the extrapolation suggested in [75] to higher dimensional problems than those tested by the authors, actually fails. Instead, we see a crossing point at around $D = 50$ and $D = 100$, after which exactly the opposite conclusion becomes true. The Gaussian search distribution performs

Table 4.3: Ranksum Statistical test for performance comparison between Gaussian and Cauchy search distribution on the Shifted Rosenbrock function with Budget = $10000 \cdot D$ and Population size = 300.

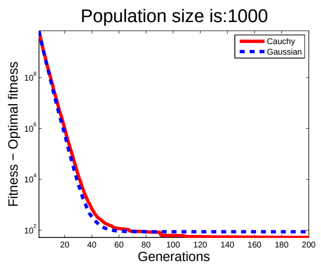
Dimension	Cauchy		Gaussian		Ranksum Test	
	mean	std	mean	std	H	P-Value
20	**9.33E+03**	**5.23E+04**	1.98E+04	6.96E+04	1	7.13E-12
30	**2.56E+04**	**2.43E+05**	3.36E+05	5.99E+05	1	8.40E-31
40	**61.8629**	**31.6877**	1.28E+06	2.81E+06	1	2.56E-34
50	**61.5353**	**25.0485**	1.56E+06	2.32E+06	1	2.56E-34
100	9.27E+07	4.62E+07	**9.41E+06**	**1.04E+07**	1	1.62E-32
200	1.84E+11	1.19E+10	**2.74E+07**	**1.95E+07**	1	2.56E-34
300	6.46E+11	3.07E+10	**6.54E+07**	**3.54E+07**	1	2.56E-34
400	1.34E+12	6.06E+10	**1.47E+08**	**7.21E+07**	1	2.56E-34
500	2.15E+12	9.43E+10	**2.71E+08**	**9.92E+07**	1	2.56E-34
1000	7.68E+12	1.78E+11	**2.12E+09**	**4.47E+08**	1	7.07E-18

Table 4.4: Ranksum Statistical test for performance comparison between Gaussian and Cauchy search distribution on the Shifted Rosenbrock function with Budget = $10000 \cdot D$ and Population size = 1000

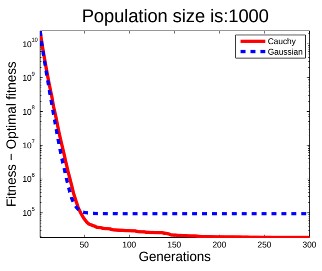
Dimension	Cauchy		Gaussian		Ranksum Test	
	mean	std	mean	std	H	P-Value
20	**52.211**	**273.2988**	87.3066	257.3303	1	1.59E-15
30	**1.87E+04**	**1.75E+05**	9.42E+04	1.93E+05	1	2.06E-29
40	**7.74E+03**	**6.41E+04**	1.24E+05	2.61E+05	1	1.13E-31
50	**1.88E+03**	**1.10E+04**	1.07E+05	1.38E+05	1	2.44E-32
100	**1.44E+04**	**8.38E+04**	4.18E+05	5.81E+05	1	1.47E-31
200	7.17E+10	6.20E+09	**9.78E+05**	**8.20E+05**	1	2.56E-34
300	4.72E+11	2.38E+10	**2.27E+06**	**1.57E+06**	1	2.56E-34
400	9.78E+11	4.11E+10	**4.86E+06**	**2.53E+06**	1	2.56E-34
500	1.56E+12	6.19E+10	**7.34E+06**	**4.01E+06**	1	2.56E-34
1000	5.25E+12	2.18E+11	**2.82E+07**	**1.05E+07**	1	7.07E-18

Table 4.5: Ranksum Statistical test for performance comparison between Gaussian and Cauchy search distribution on the Shifted Rosenbrock function with Budget = $10000 \cdot D$ and Population size = 2000.

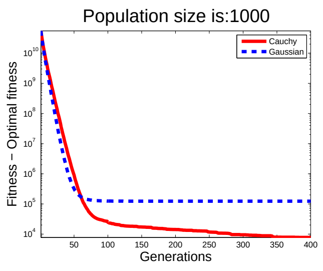
Dimension	Cauchy		Gaussian		Ranksum Test	
	mean	std	mean	std	H	P-Value
20	**20.3619**	**21.5225**	23.8297	31.5721	1	3.12E-24
30	**1.15E+04**	**4.95E+04**	5.30E+04	9.35E+04	1	2.13E-21
40	**8.03E+03**	**6.32E+04**	5.18E+04	5.40E+04	1	3.45E-30
50	**338.7745**	**976.4567**	5.66E+04	7.39E+04	1	1.54E-33
100	**8.04E+03**	**4.76E+04**	1.55E+05	1.65E+05	1	3.56E-32
200	5.44E+10	5.85E+09	**2.31E+05**	**1.93E+05**	1	2.56E-34
300	4.53E+11	2.01E+10	**3.88E+05**	**2.79E+05**	1	2.56E-34
400	9.18E+11	4.34E+10	**7.72E+05**	**4.38E+05**	1	2.56E-34
500	1.35E+12	4.89E+10	**1.14E+06**	**6.06E+05**	1	2.56E-34
1000	3.77E+12	1.29E+11	**5.03E+06**	**1.66E+06**	1	7.07E-18



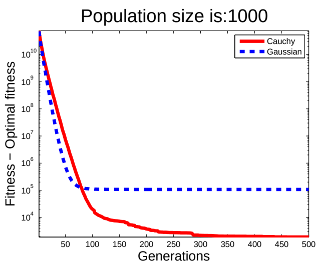
(a) Dimension 20



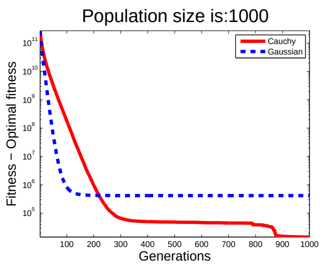
(b) Dimension 30



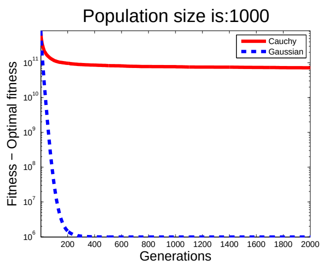
(c) Dimension 40



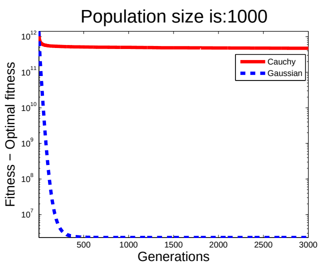
(d) Dimension 50



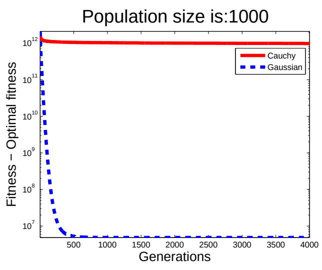
(e) Dimension 100



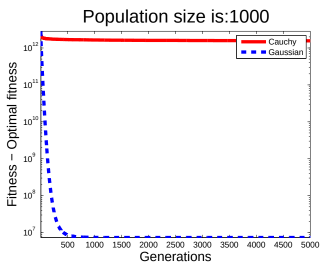
(f) Dimension 200



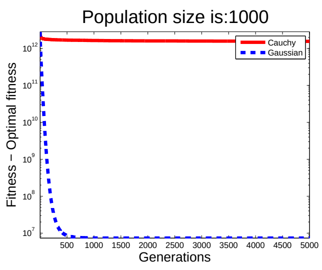
(g) Dimension 300



(h) Dimension 400



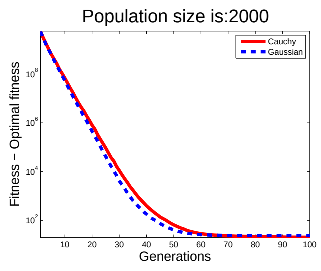
(i) Dimension 500



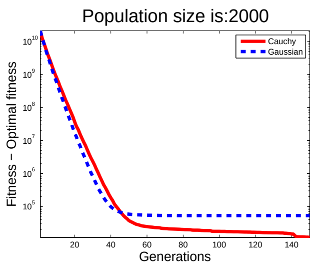
(j) Dimension 1000

Figure 4.2: Convergence behaviour of Gaussian and Cauchy as search distribution for EDA-MCC on different dimensions of the Rosenbrock function for population size of 1000 on average, as obtained from 25 independent runs. A total budget of $1 \cdot 10^4 \cdot D$, where D is the dimension of the problems were used.

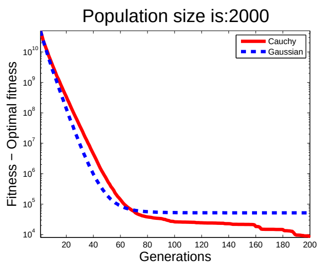
significantly better than the Cauchy at problem dimensions larger than $D = 100$, up to $D = 1000$.



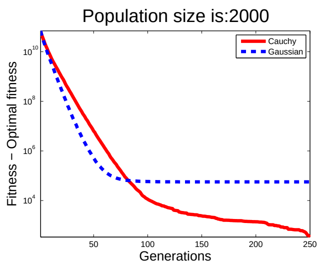
(a) Dimension 20



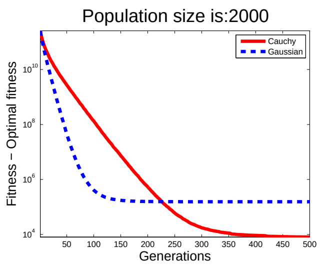
(b) Dimension 30



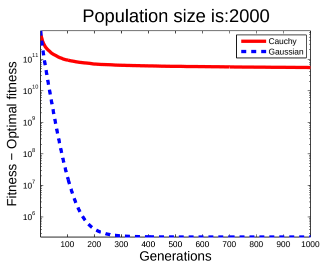
(c) Dimension 40



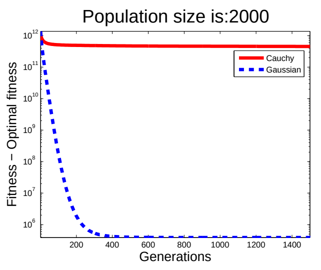
(d) Dimension 50



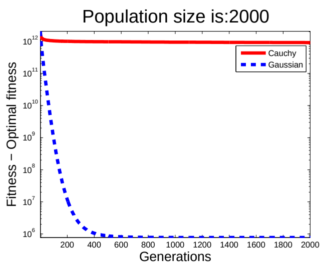
(e) Dimension 100



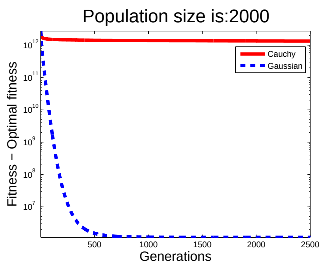
(f) Dimension 200



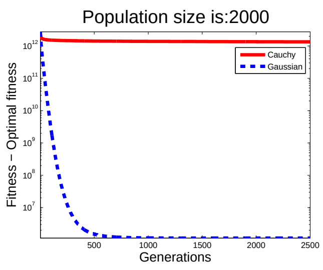
(g) Dimension 300



(h) Dimension 400



(i) Dimension 500



(j) Dimension 1000

Figure 4.3: Convergence behaviour of Gaussian and Cauchy as search distribution for EDA-MCC on different dimensions of the Rosenbrock function for population size of 2000 on average, as obtained from 25 independent runs. A total budget of $1 \cdot 10^4 \cdot D$, where D is the dimension of the problems were used.

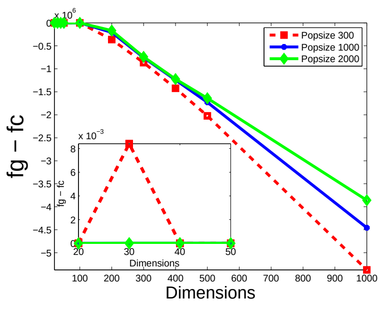
We found the above conclusion consistently (up to slight shifts of the crossing point) with different population sizes. This will be apparent in the next subsection where complete and summary plots of results obtained with three different population sizes will be presented. Moreover, as we shall see, the finding that Gaussian performs better than Cauchy in high (beyond 100) dimensional problems is also observed for all benchmark problems tested.

4.4.2 Results of an extensive empirical study

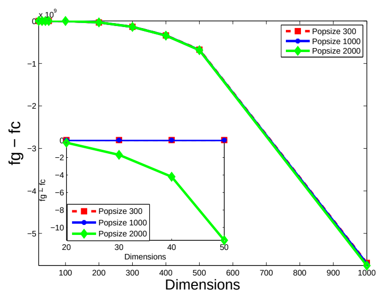
Having found an interesting pattern of comparative behavior in the previous section on the shifted Rosenbrock function, we then performed similar comparative experiments on all functions from Table 4.1 in order to see if our finding holds more generally.

Figure 4.4 presents all these results in a compact format. Here we display the differences between the fitness value achieved with Gaussian (fg) and with Cauchy (fc) search distributions respectively. By fitness value we mean the average of the best fitness in the last generation, as averaged over 25 independent runs. Whenever this difference ($fg - fc$), is positive it means that Cauchy outperformed Gaussian (recall, we do minimization so smaller fitness is better), and vice-versa – whenever ($fg - fc$) is negative then Gaussian outperformed Cauchy. The 7 plots correspond to the 7 benchmark problems tested, and each curve on these plots corresponds to a particular choice of population size. Since the fitness differences are much larger when D is large, we also show a zoomed version of the lower dimensional regime in order to better see the details.

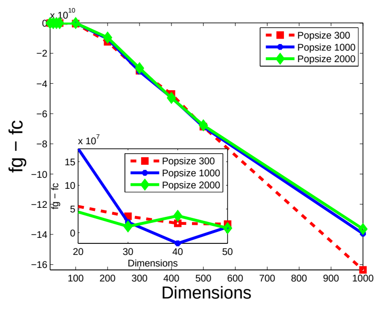
From Figure 4.4, we see that the comparative behaviour of the two search distributions in the high dimensional regime, as observed in the previous section, consistently holds up on all functions tested, and with all population sizes tested. That is, the differences in the fitness values ($fg - fc$) are positive in the dimension range 20-50 in most cases, meaning that Cauchy tends to be better in this regime. But, as the dimension exceeds 50 or 100, the differences become negative and remain negative, indicating that Gaussian is now better than Cauchy.



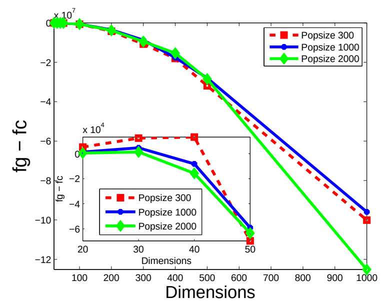
(a) Shifted Sphere Function



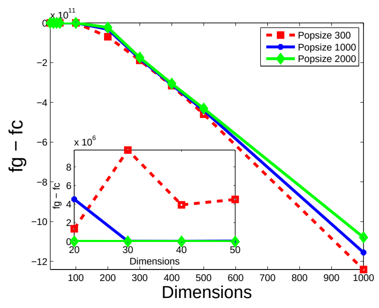
(b) Shifted Schwefel's Problem 1.2



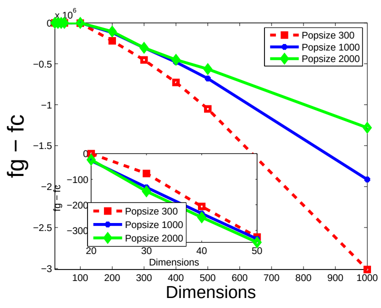
(c) Shifted rotated Elliptic Function



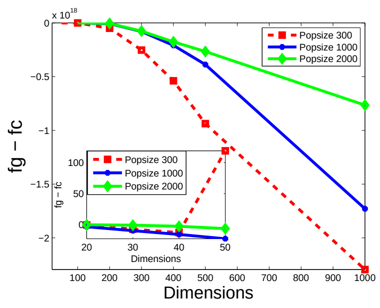
(d) Shifted Schwefel's Problem 1.2 With Noise in fitness



(e) Shifted Rosenbrock Function



(f) Shifted Rastrigin Function



(g) Exp. Ext. Griewank Fn + Rosenbrock

Figure 4.4: Differences between the average (from 25 repeated runs) of the best fitness values achieved by the Cauchy (fc) and by the Gaussian (fg) EDAs, as the dimension is varied, for seven test problems. The smaller plots superimposed represent zoomed versions of the same results in the range of 20-50 dimensions.

We can also see from figure 4.4 that the results with smaller population size yield the largest contrast between the performances of these two search distributions. We therefore conclude on the basis of these results that Cauchy may be better than Gaussian in low dimensional problems, but Gaussian is superior in high dimensional problems. Statistical tests confirmed that these differences are statistically significant.

4.4.3 Further results when the optimum is shifted much further away

Since Cauchy sampling in optimisation is expected to have an advantage over Gaussian when long jumps are beneficial, we also tried to modify the test problems by shifting the global optimum and increasing the search box sizes from $[-10^2 \;\; 10^2]$ up to $[-10^4 \;\; 10^4]$, to see if Cauchy's long jumps will pay off. We found this is not the case, and Cauchy search makes very slow progress in all cases tested. See results in Figures 4.5 and 4.6. These experiments conclude that Cauchy's long jumps do not help in high dimensions, which agrees with the findings in [38]. That is, the chances for a long jump to turn out lucky vanish with increasing dimension, and the analysis in the next section suggests that in fact this issue is unavoidable.

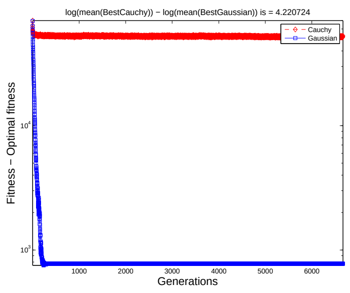
4.5 Understanding the reasons for our experimental findings

We attack the problem from two different angles. First, we look at the probability of bad moves. Second, we examine the behaviour of Gaussian and Cauchy norms as the dimensionality increases.

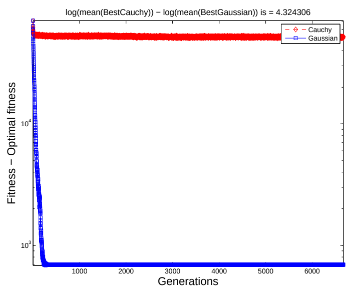
4.5.1 Probability of bad moves

Here, we show why large search steps are, in general, more likely to perform worse than smaller ones and explain the role that the problem dimensionality plays in this issue.

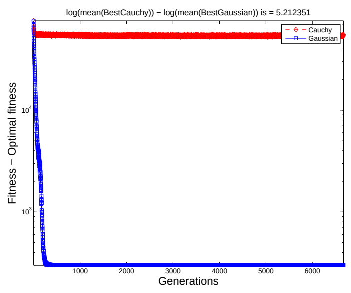
We start by considering a search distribution that selects new candidate solutions from the uniform distribution on a sphere of fixed radius, r, about a current population



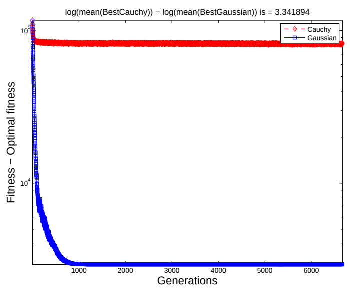
(a) Rosenbrock; Shift = Default



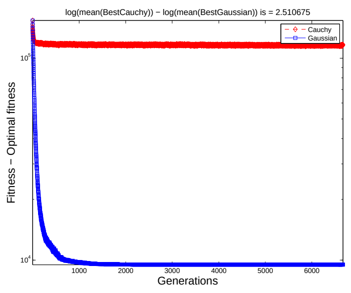
(b) Rosenbrock; Shift = 100



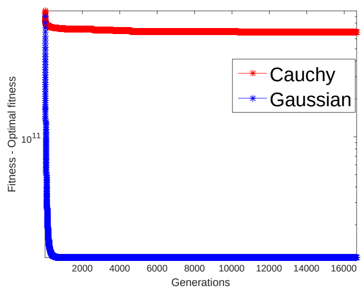
(c) Rosenbrock; Shift = 1000



(d) Rosenbrock; Shift = 5000



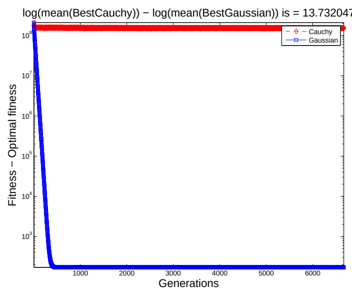
(e) Rosenbrock; Shift = 8000



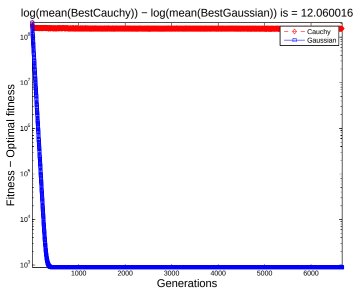
(f) Rosenbrock; Shift = 10000

Figure 4.5: Trajectories to compare Gaussian and Cauchy search distributions on the Shifted Rosenbrock Function when the box size was increased from $[-10^2 \;\; 10^2]$ to $[-10^4 \;\; 10^4]$ and the shift values varied from $10^2 - 10^4$.

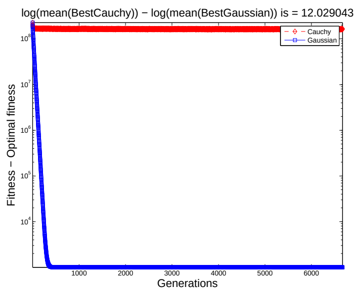
member – why this captures the essential behaviour of Gaussian high-dimensional search will be explained shortly – and we look at the effect of varying r. More precisely we consider the probability of the event that a new candidate solution is closer to the global (or any particular local) optimum than the current population member. See Figure 4.7 – the point $x^\star$ is the global optimum in the search space, the point $p^\star$ is the centre (mean) of the current population, and the shaded circle represents the ball of radius $\rho := \|x^\star - p^\star\|$ centred on $x^\star$. Clearly a new candidate solution p' is closer to the global optimum than $p^\star$ if and only if it lies within this ball, that is when $\|x^\star - p'\| < \rho$. In Figure 4.7 we see this intersection in bold for several choices of r – in 2 dimensions this intersection is an arc, in 3 it is a spherical cap, and in 4 or more dimensions it is a hyperspherical cap. Now, what is the probability of the event $\|x^\star - p'\| < \rho$? Denote by S_r^{D-1} the



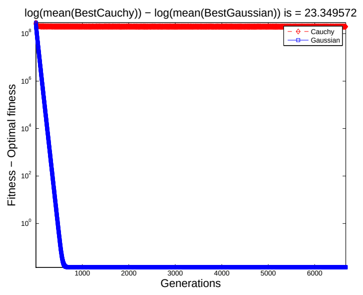
(a) Sphere; Shift = 3000



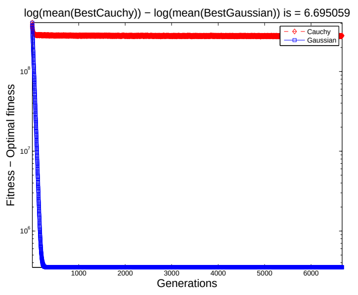
(b) Sphere; Shift = 300000



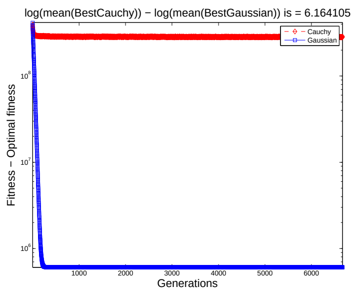
(c) Sphere; Shift = 3000000



(d) Sphere; Shift = 10000000



(e) Sphere; Shift = 20000000



(f) Sphere; Shift = 20500000

Figure 4.6: Comparisons of Gaussian vs. Cauchy search distributions on problems with highly shifted optima and increased sizes of the search box.

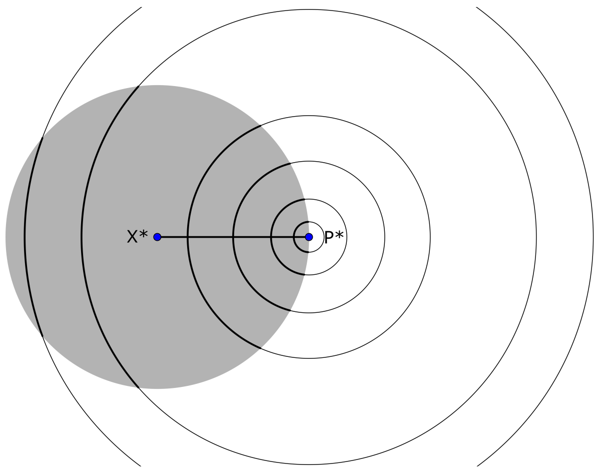


Figure 4.7: Proof by picture – the probability that $\|x^\star - p'\| < \|x^\star - p^\star\|$ is monotonically decreasing in the step size of the search.

sphere about $p^\star$ of radius r in $\mathbb{R}^D$: When p' is drawn from the uniform distribution on S_r^{D-1}, this probability is the proportion of the surface of the whole sphere comprising the intersection, namely the quotient of the surface area of the hyperspherical cap to the sphere S_r^{D-1}. For a fixed value of $\|x^\star - p^\star\|$, and for any problem dimensionality $D \geqslant 2$, this probability is monotonically decreasing in r for $r \in (0, 2\rho)$[1] and, of course, it is zero for values of $r > 2\rho$ in *any* dimension. Thus if the search direction from a current solution is chosen uniformly at random then, irrespective of any other consideration, larger step sizes are always more likely to take us further from the global optimum than smaller step sizes. How fast does this probability decay as a function of the step size or of the dimensionality? Define the angle of the hyperspherical cap at $p^\star$ to be $2\theta_r$, and note that the proportion of the sphere of radius r covered by this cap is the same as the proportion of the unit sphere covered by a cap on the unit sphere also with angle $2\theta_r$. Therefore $\Pr\{\|x^\star - p'\| < \rho\} \leqslant \exp(-\frac{D^2}{2}\cos^2\theta_r)$ where the RHS follows from Lemma 2.2 of [7] which upper bounds this latter quantity. By simple trigonometry one finds that $\cos\theta_r = r/2\rho$, and thus we obtain the following theorem:

Theorem 4.5.1 (Most Search Steps are Bad). Let $x^\star$, $p^\star$ be two fixed points in $\mathbb{R}^D$ with the Euclidean distance between them $\rho := \|x^\star - p^\star\|$. Let $p' = p^\star + z$ where z is sampled from the uniform distribution on the hypersphere of radius r. Then:

$$\Pr\left\{\|x^\star - p'\| > \|x^\star - p^\star\|\right\} > 1 - \exp\left(-\frac{D^2 r^2}{8\rho^2}\right) \tag{4.1}$$

$\square$

This means that, for any fixed value of ρ, the probability of sampling a point closer to the global optimum than the current reference point decays exponentially quickly in both the search radius (step size) r, and the dimensionality D. It also means that, for any choice of relative step size r/ρ, the proportion of good directions (i.e. directions that get us closer to the optimum than the reference point) decays exponentially quickly in the problem dimension. Therefore, if the step direction is random, large steps in high-dimensional search spaces are far less likely to take us closer to the global optimum

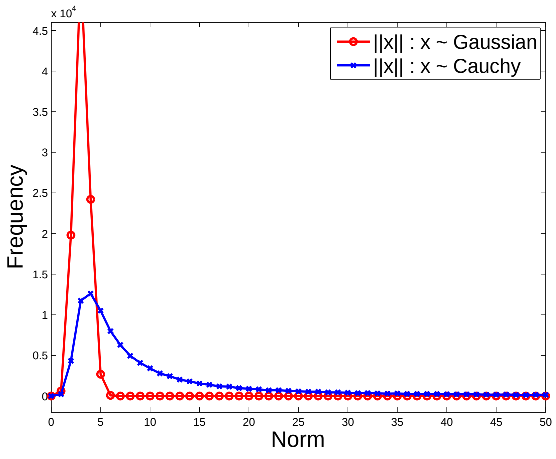
[1]In dimension 1 this probability is exactly 0.5 for a step of size $r \in (0, 2\rho)$.

than small steps, and thus for high-dimensional search we would expect that with very high probability heavy-tailed distributions such as the Cauchy will perform poorly. This suggests that exploration by large steps is mostly counterproductive in high dimensions and instead one should focus mainly on finding the right direction in which to move the search distribution.

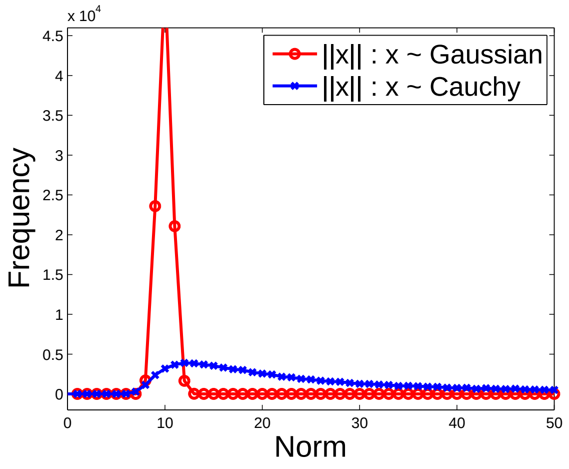
4.5.2 Gaussian and Cauchy norms in high dimensions

Now we discuss some possible reasons why a Gaussian search distribution does better. From high dimensional probability theory it is known that high dimensional probability distributions may look very different from their low dimensional versions, and may therefore behave in a counter-intuitive manner. We conjecture the good performance of the Gaussian search may be due to its good concentration property, which the Cauchy distribution lacks. This property means that in high dimensions most of the points sampled from the distribution lie within a *thin shell* at some distance from the center of the distribution - in other words, in high dimensions we will not generate new points very close to the mean, neither will we generate points very far from the mean either. Figure 4.8 demonstrates this empirically. We sampled 100,000 points from a 10, 100, 200 and 1000-dimensional standard Gaussian and plotted the histogram of Euclidean distances from the origin (centre of the distribution). We see from the figure that all of these distances are close to approximately $\sqrt{D}$ ($\sqrt{10} = 3.16$, $\sqrt{100} = 10$, $\sqrt{200} = 14.14$, $\sqrt{1000} = 31.66$). This was just an example for figure 4.8, but in general in the algorithm we use $\sqrt{\text{Tr}(\Sigma)}$. So, as the dimensionality increases we have most of the points within a shell that gets thinner and thinner relative to the average distance from the centre.

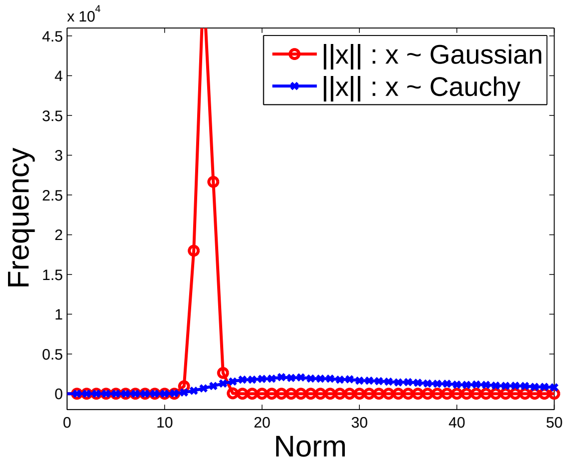
We then repeated the same experiment with 10, 100, 200 and 1000-dimensional Cauchy norms where 70% of the components of the points were sampled from independent c-



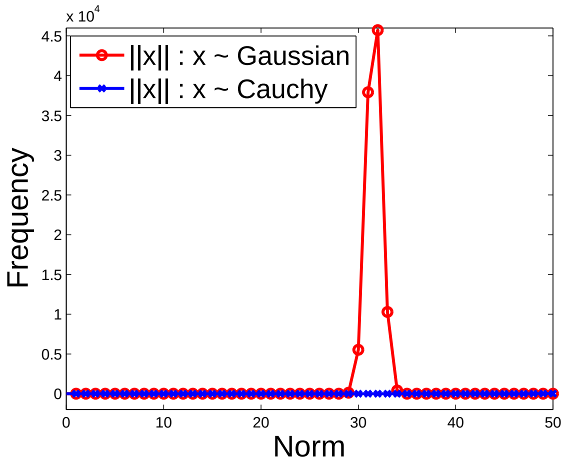
(a) D = 10, c = 2



(b) D = 100, c = 20



(c) D = 200 , c = 20



(d) D = 1000 , c = 20

Figure 4.8: Comparison of the histograms of Gaussian vs. Cauchy norms as D increases. The values of the parameter c chosen here (i.e. the dimension of independent multivariate Cauchy components) correspond to a population size of 300 (although we observed no qualitative difference for other choices). We used 100,000 sample points to create these histograms.

dimensional multivariate standard Cauchy distributions and the remaining 30% from independent standard Gaussian – this mimics a typical SM & WI split from our Cauchy-EDA-MCC simulations. We superimposed these histograms on the same plots with the Gaussian norms in Figure 4.8. From Figure 4.8 it is very apparent that the Gaussian norms are all clamped in a narrow range, whereas the Cauchy norms are increasingly spread out. This will have implications on the implicit searching strategy associated with these two distributions, as we shall discuss in the remainder of this section.

Take the Gaussian case first. More formally, for a generic non-degenerate[1] $D \times D$ covariance matrix Σ, let $X \sim N(0, \Sigma)$. Then the expected norm can be approximated as follows:

$$E[||X||] \leqslant \sqrt{E[||X||^2]} = \sqrt{\mathrm{Tr}(\Sigma)} \tag{4.2}$$

using Jensen's inequality. Indeed, applying the linearity of expectation, we have

$$E[||X||^2] = E[\sum_{i=1}^{D} X_i^2] = \sum_{i=1}^{D} E[X_i^2] = \sum_{i=1}^{D}(\Sigma_{ii}) = \mathrm{Tr}(\Sigma).$$

Note that in the case $\Sigma = I$ we have $\sqrt{\mathrm{Tr}(\Sigma)} = \sqrt{D}$. This is why we saw the averages of Gaussian norms at approximately $\sqrt{D}$ in Figure 4.8. Furthermore, the following lemma shows that with high probability $\|X\|$ is close to $\sqrt{\mathrm{Tr}(\Sigma)}$ (in absolute difference relative to the spectral norm of Σ).

Lemma 2. Let $X \in R^D$ where X has entries drawn from a multivariate Gaussian with

[1]Note that the model complexity control on the covariance estimates in EDA-MCC ensures that the covariance estimates are indeed non-degenerate – of course, provided that we set the parameters c and m wisely (as discussed in an earlier section).

mean zero and Σ covariance. Then, $\forall \varepsilon \in (0,1)$,

$$Pr\left\{\left| \|X\| - \sqrt{\mathrm{Tr}(\Sigma)} \right| \geqslant \varepsilon \sqrt{\lambda_{\max}(\Sigma)} \right\} \leqslant 2\exp\left[-\frac{\varepsilon^2}{2}\right] \tag{4.3}$$

$\square$

Lemma 2 implies that in Gaussian EDA search, a large fraction of the new generation lies in a thin shell at the same distance from the center of the population – therefore selection of the fittest points essentially selects the promising *directions*. These two elements – using all of the available resources to select directions, and then ensuring a steady move of size just below $\sqrt{\mathrm{Tr}(\Sigma)}$ from the center of the population from one generation to the next – provide Gaussian EDA a well focused strategy that is beneficial and resource-efficient. And of course, as we approach a local optimum $\mathrm{Tr}(\Sigma)$ will decay, so in fact Gaussian EDA automatically tunes the search granularity over successive generations.

By contrast, the Cauchy density does not have good concentration properties. This is very apparent from the numerical experiment in Figure 4.8. While we see a reasonably high density region in the case of $D = 10$, as D increases, the heavy tails of the distribution in all directions dissolve any high density region. Therefore, Cauchy based search has no ability to prioritize selecting good directions.

This probability inequality was mentioned in [45] without proof. Here we derive it from Lemma 1 of [25].

Proof: The following bounds [25] hold for the Gaussian square norm, with the two sides holding with different probabilities.

$$Pr\left\{ \|X\| \geqslant \sqrt{(1+\varepsilon)Tr(\Sigma)} \right\} \leqslant \exp\left(-\frac{Tr(\Sigma)(\sqrt{1+\varepsilon}-1)^2}{2\lambda_{\max}(\Sigma)} \right) \tag{4.4}$$

$$Pr\left\{ \|X\| \leqslant \sqrt{(1-\varepsilon)Tr(\Sigma)} \right\} \leqslant \exp\left(-\frac{Tr(\Sigma)(\sqrt{1-\varepsilon}-1)^2}{2\lambda_{\max}(\Sigma)} \right) \tag{4.5}$$

Here we massage this pair of bounds into single bound.

Now from the Left Hand Side (LHS) of equation 4.4 and our target, we set:

$$\sqrt{Tr(\Sigma)+\varepsilon Tr(\Sigma)} = \sqrt{Tr(\Sigma)} + \tau\sqrt{\lambda_{\max}(\Sigma)}$$

Squaring both sides and expanding the Right Hand Side (RHS) expression we have

$$\varepsilon Tr(\Sigma) = \tau\left[2\sqrt{Tr(\Sigma)\lambda_{\max}(\Sigma)} + \tau\lambda_{\max}(\Sigma) \right]$$

$$\varepsilon = \tau\left[\frac{2\sqrt{Tr(\Sigma)\lambda_{\max}(\Sigma)} + \tau\lambda_{\max}(\Sigma)}{Tr(\Sigma)} \right]$$

Solving for ε, and replacing it

$$\varepsilon = \tau\left[2\sqrt{\frac{\lambda_{\max}(\Sigma)}{Tr(\Sigma)}} + \frac{\tau\lambda_{\max}(\Sigma)}{Tr(\Sigma)} \right]$$

Substituting ε into the RHS of eq.4.4 gives, after some algebra:

$$\exp\left[-\frac{Tr(\Sigma)}{2}\left(\sqrt{\frac{1}{\lambda_{\max}(\Sigma)}+\frac{2\tau}{\sqrt{Tr(\Sigma)\lambda_{\max}(\Sigma)}}+\frac{\tau^2}{Tr(\Sigma)}}-\frac{1}{\sqrt{\lambda_{\max}(\Sigma)}}\right)^2\right]$$

Taking the Lowest Common Multiple (LCM) of the term under the square root, we have

$$=\exp\left[-\frac{Tr(\Sigma)}{2}\left(\sqrt{\frac{Tr(\Sigma)+2\tau\sqrt{Tr(\Sigma)\lambda_{\max}(\Sigma)}+\tau^2\lambda_{\max}(\Sigma)}{Tr(\Sigma)\lambda_{\max}(\Sigma)}}-\frac{1}{\sqrt{\lambda_{\max}(\Sigma)}}\right)^2\right]$$

$$=\exp\left[-\frac{Tr(\Sigma)}{2}\left(\sqrt{\frac{(\sqrt{Tr(\Sigma)}+\tau\sqrt{\lambda_{\max}(\Sigma)})^2}{Tr(\Sigma)\lambda_{\max}(\Sigma)}}-\frac{1}{\sqrt{\lambda_{\max}(\Sigma)}}\right)^2\right]$$

$$=\exp\left[-\frac{Tr(\Sigma)}{2}\left(\frac{\sqrt{Tr(\Sigma)}+\tau\sqrt{\lambda_{\max}(\Sigma)}}{\sqrt{Tr(\Sigma)\lambda_{\max}(\Sigma)}}-\frac{1}{\sqrt{\lambda_{\max}(\Sigma)}}\right)^2\right]$$

Now taking LCM of the term inside the square, and

$$=\exp\left[-\frac{Tr(\Sigma)}{2}\left(\frac{\sqrt{Tr(\Sigma)}+\tau\sqrt{\lambda_{\max}(\Sigma)}-\sqrt{Tr(\Sigma)}}{\sqrt{Tr(\Sigma)\lambda_{\max}(\Sigma)}}\right)^2\right]$$

simplifying, we get:

$$=\exp\left[-\frac{Tr(\Sigma)}{2}\left(\frac{\tau\sqrt{\lambda_{\max}(\Sigma)}}{\sqrt{Tr(\Sigma)\lambda_{\max}(\Sigma)}}\right)^2\right]$$

after cancellations. Rename τ by ε, and this completes the proof for one side of Lemma

1. The other side is analogous, and yields:

$$Pr\left\{ \|X\| - \sqrt{Tr(\Sigma)} \leqslant -\varepsilon\sqrt{\lambda_{\max}(\Sigma)} \right\} \leqslant \exp\left[-\frac{\varepsilon^2}{2} \right]$$

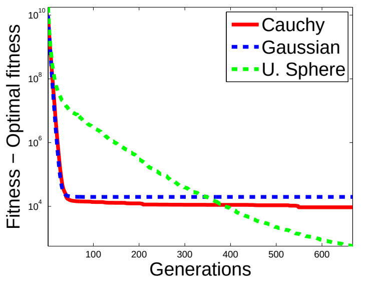
□

In the sequel we shall put the above explanation to a test: We shall create a new search distribution for EDA that takes to the extreme the clever implicit searching strategy of Gaussian EDA that we just uncovered. If our reasoning above is correct, then the new search distribution might perform even better in high dimensions.

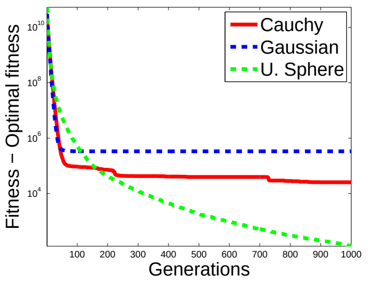
4.6 EDA with uniform search distribution on a hypersphere

Rather than searching in a thin shell at some constant distance from the center of the population, let us search precisely on the hypersphere with the same radius. Based on our analysis in the previous section, from eqs. (4.2)-(4.3), we define the search distribution as a uniform distribution on the sphere of radius $\sqrt{\mathrm{Tr}(\Sigma)}$, where, as before, Σ is the covariance estimated from the selected individuals. This way, when the high fitness individuals are selected they represent exactly the high fitness directions at granularity equal to the radius. The subsequent generation then makes a steady move towards the average of the selected directions, just like it was the case for Gaussian based search.

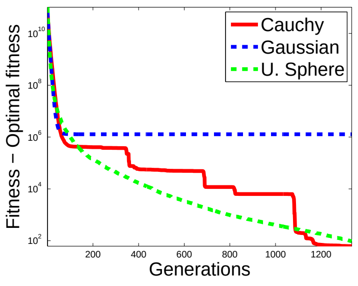
We tested and validated the performance of this new EDA variant in an extensive series of experiments, comparatively with both the Gaussian and the Cauchy EDA variants discussed earlier. We first present detailed results on the search process for the Shifted Rosenbrock function in Figures 4.9 - 4.11, for three different population sizes, each tested on all different dimensions of the problem considered, from low to high. We also present a summary of the comparative results in figure 4.12. As conjectured, we can see that the uniform sphere based search strategy becomes increasingly efficient in high dimensions and outperforms both Cauchy and Gaussian based EDA search as the dimensionality of the problem increases. We confirmed using ranksum tests that these differences are statistically significant. This is because in an exponentially increasing search space, when only having a linearly increasing budget it becomes more and more important to prioritize the task of selecting good directions. We also see that this effect is very robust and not influenced by the particular choice of population size. All plots represent average of best fitness as computed from 25 independent runs. The total budget was set to $10^4 \cdot D$, where D is the dimension of the problems.



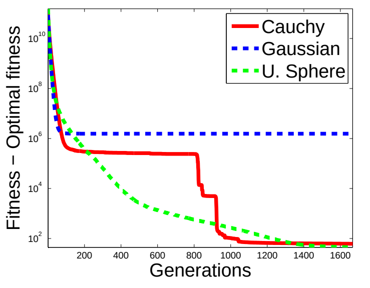
(a) Dimension 20



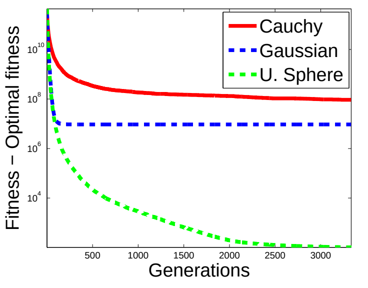
(b) Dimension 30



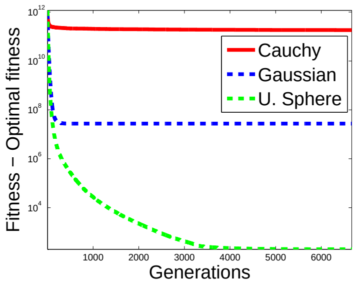
(c) Dimension 40



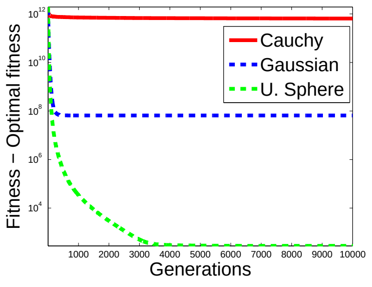
(d) Dimension 50



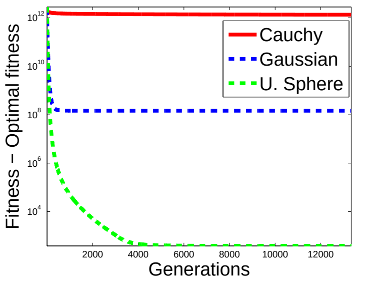
(e) Dimension 100



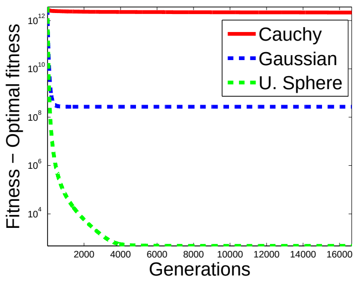
(f) Dimension 200



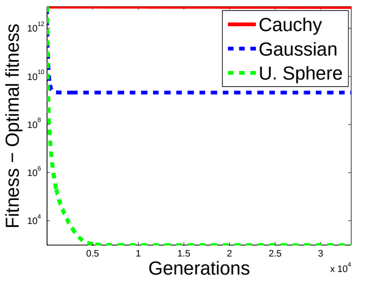
(g) Dimension 300



(h) Dimension 400

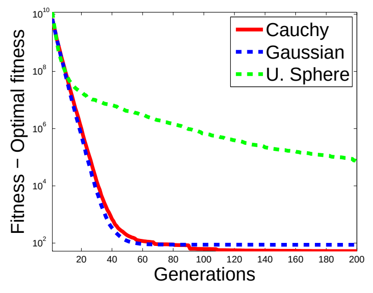


(i) Dimension 500

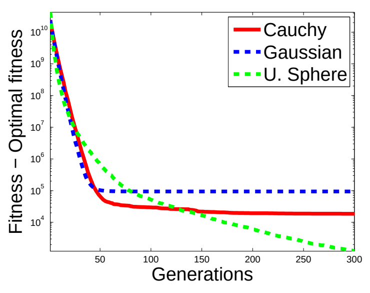


(j) Dimension 1000

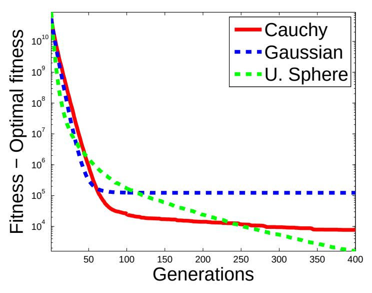
Figure 4.9: Trajectories of Gaussian EDA-MCC, Cauchy EDA-MCC and UniformSphere EDA on the Shifted Rosenbrock function when the population size is $N = 300$.



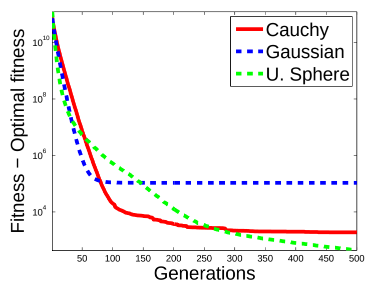
(a) Dimension 20



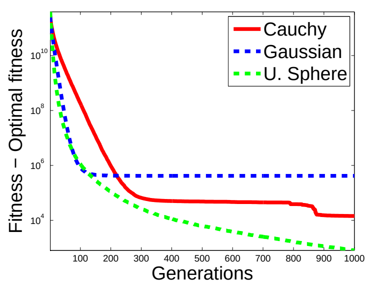
(b) Dimension 30



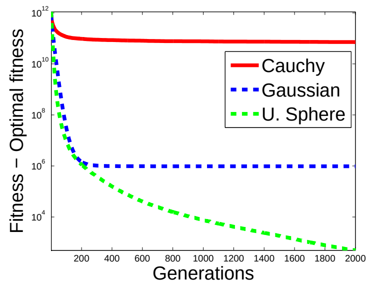
(c) Dimension 40



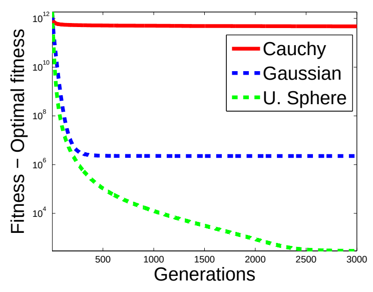
(d) Dimension 50



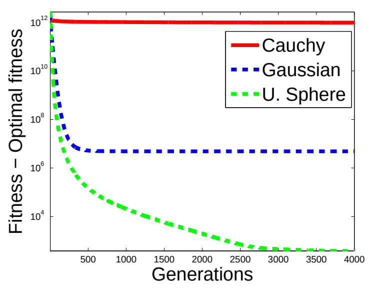
(e) Dimension 100



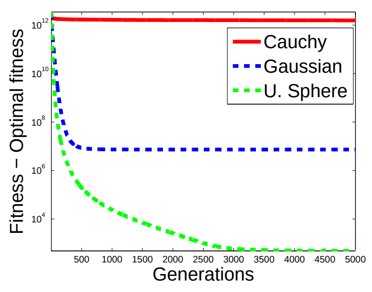
(f) Dimension 200



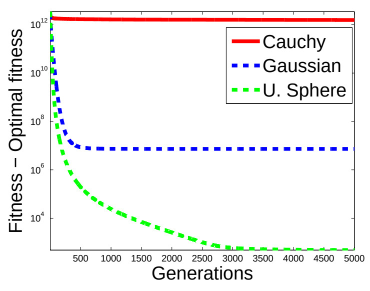
(g) Dimension 300



(h) Dimension 400

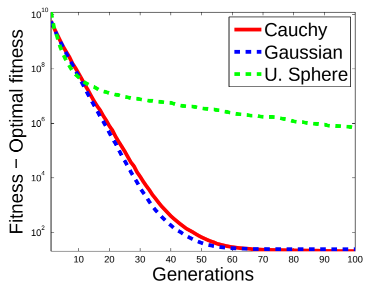


(i) Dimension 500

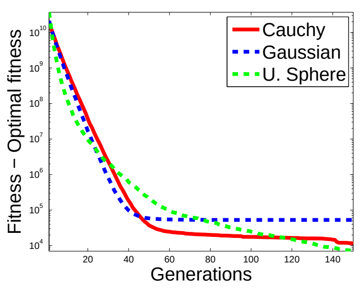


(j) Dimension 1000

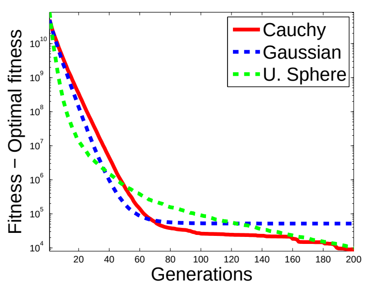
Figure 4.10: Trajectories of Gaussian EDA-MCC, Cauchy EDA-MCC and UniformSphere EDA on the Shifted Rosenbrock function when the population size is $N = 1000$.



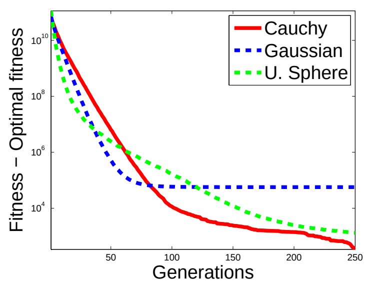
(a) Dimension 20



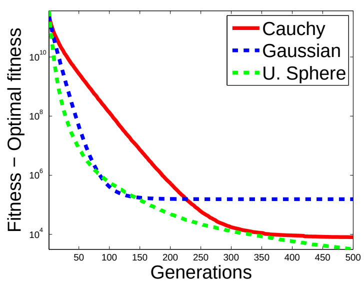
(b) Dimension 30



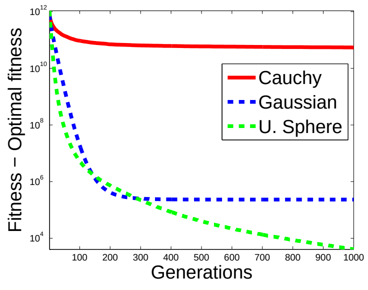
(c) Dimension 40



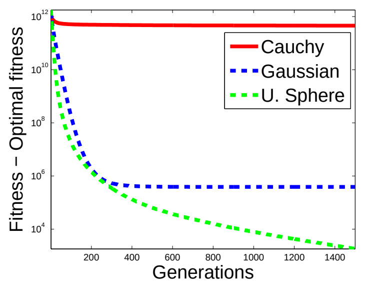
(d) Dimension 50



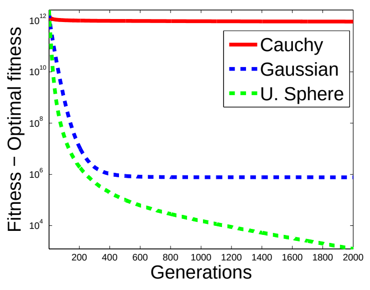
(e) Dimension 100



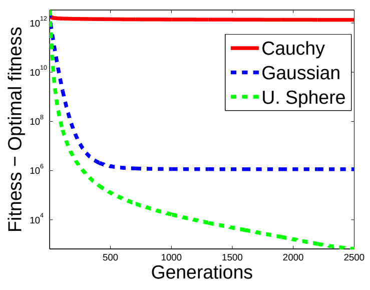
(f) Dimension 200



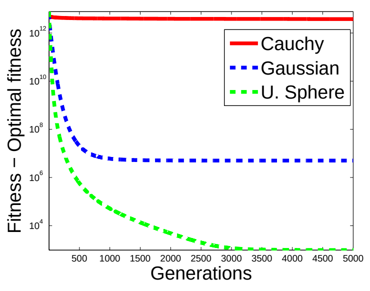
(g) Dimension 300



(h) Dimension 400

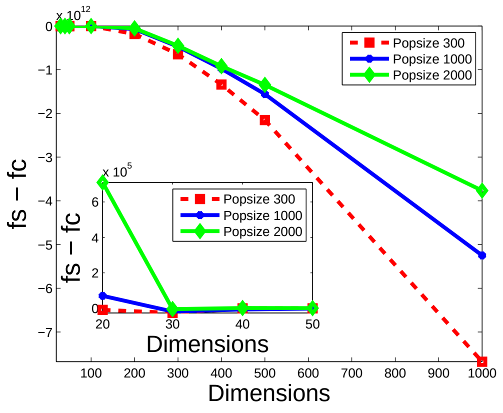


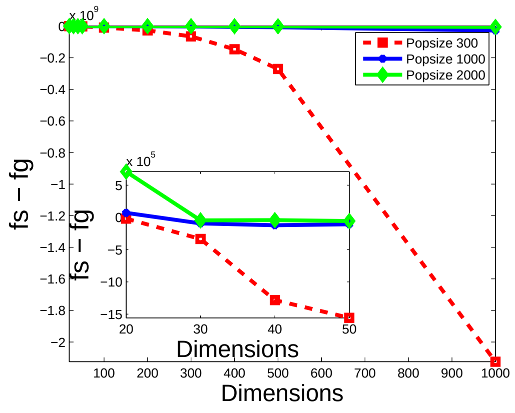
(i) Dimension 500

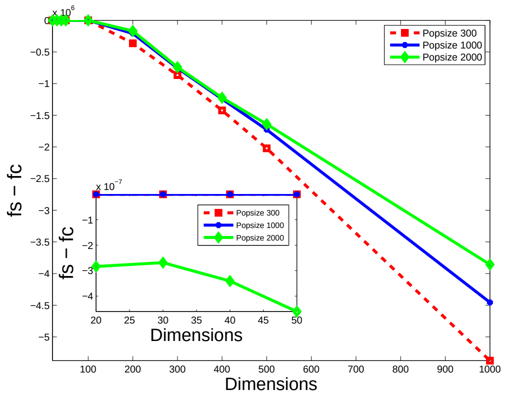


(j) Dimension 1000

Figure 4.11: Trajectories of Gaussian EDA-MCC, Cauchy EDA-MCC and UniformSphere EDA on the Shifted Rosenbrock function when the population size is $N = 2000$.


(a) Uniform Sphere vs. Cauchy for Shifted Rosenbrock function


(b) Uniform Sphere vs. Gaussian for Shifted Rosenbrock function


(c) Uniform Sphere vs. Cauchy for Shifted Sphere function

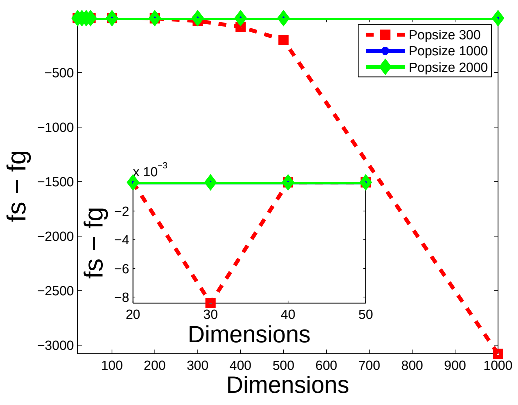
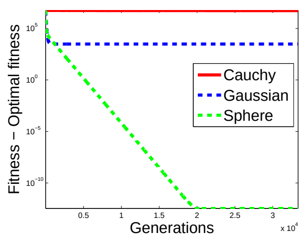
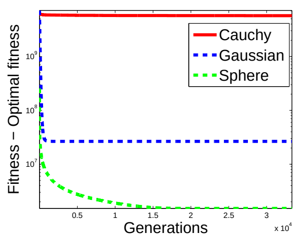

(d) Uniform Sphere vs. Gaussian for Shifted Sphere function

Figure 4.12: Differences between the averages (from 25 repeated runs) of the best fitness values achieved by the Gaussian (fg) and Uniform on Sphere (fs) plotted on the right and Cauchy (fc) and Uniform on Sphere (fs) EDAs plotted on the left, as the dimension is varied. The smaller plots superimposed represent zoomed versions of the same results in the range of 20-50 dimensions.

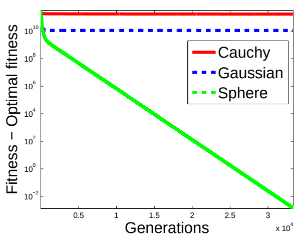
Finally, in Figure 4.13 we demonstrate the results of large scale experiments in 1000-dimensions on the remaining 6 benchmark function listed in Table 4.1. Here we used a population size of $N = 300$. Again we see that UniformSphere-EDA consistently and significantly outperforms the other two EDA variants.



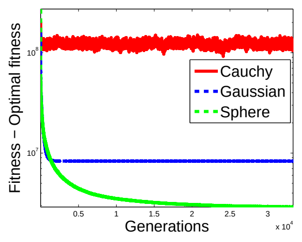
(a) Shifted Sphere Function



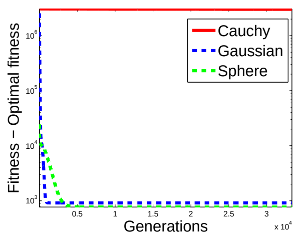
(b) Shifted Schwefel's Problem 1.2



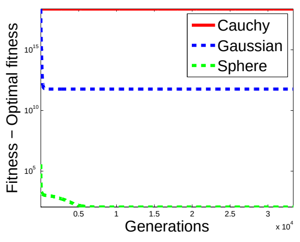
(c) Shifted Rotated High Conditioned Elliptic Function



(d) Shifted Schwefel's Problem 1.2 with Noise in Fitness



(e) Shifted Rastrigin's Function



(f) Expanded Extended Griewank's plus Rosenbrock's Function (F8F2)

Figure 4.13: Comparison of Gaussian EDA-MCC, Cauchy EDA-MCC and UniformSphere-EDA on 1000-dimensional problems. The population size was 300, and each curve is the average of the best fitness values from 25 independent runs. The budget of function evaluations was $10^4 \cdot D$, where D is the dimension of the problem.

When we artificially completely remove the deviation between the norm and the square root of the trace, this resulted in the sphere with radius square root of trace. The fact that the sphere base search works for optimisation confirms our conjecture that concentration of norms helps.

From these results, and recalling our rationale for creating this new EDA version, we conclude that our study resolved the controversy about the merits of Gaussian against Cauchy EDA search in high dimensional problems and provides a better understanding of why Gaussian performs better than Cauchy in truly high dimensional problems.

4.7 Summary

In this chapter, we conducted a large empirical study to benchmark the performance of Cauchy and Gaussian search distributions in EDA using a scalable black-box EDA optimizer. Our empirical results suggest that Cauchy search distributions perform particularly badly in high-dimensional spaces. To explain this phenomenon we developed some theory that explains why large search steps are inefficient in high dimensional search spaces. We argued that a Gaussian search distribution has an in-built prioritizing strategy that implicitly focuses resources within a generation on selecting good search directions: This strategy is a by-product of the concentration property of Gaussian norms in high dimensions. On the other hand, Cauchy norms lack good concentration properties and make a relatively high proportion of (very) large steps, and this results in an increasingly inefficient search strategy when the problem dimension increases. Based on our theoretical insights and understanding of high dimensional domains, we proposed a minor modification to the standard Gaussian EDA which enforces search within a generation to take place at a fixed radius of the current population centre. Initial experiments on a battery of test problems indicate that this simple change improves high dimensional search considerably.

CHAPTER 5

Heavy Tails with Parameter Adaptation in Random Projection based Continuous EDA

In this chapter[1], we present a new variant of EDA for high dimensional continuous optimisation, which extends a recently proposed random projections (RP) ensemble based approach by employing heavy tailed random matrices. In particular, we use random matrices with i.i.d. t-distributed entries. EDA model building has the role of capturing the structure of the problem, which in some applications may be of independent interest. The UniformSphere distribution which we proposed in Chapter 4 loses out on this aspect. Therefore in this chapter, we return to Gaussian having seen its superior performance over Cauchy in high dimensional problems in chapter 4, but with a strong regularisation implicitly implemented by an ensemble of Random Projections (RPs). This pushes the covariance matrix closer to spherical while allowing the search distribution to capture a full covariance that can be used for interpretation, i.e to learn about the structure of the problem. We generate the entries of the random matrices from a t-distributions in

[1]A shorter version of this work presented in this chapter appears in Proc. IEEE International congress on Evolutionary Computation (CEC) 2015, IEEE Xplore pp. 2074 - 2081, (c) IEEE, 2015. **Recipient of the Runner Up Student Paper Award.**

this method which may look surprising in the context of random projections, however we show that the resulting ensemble covariance is enlarged when the degree of freedom parameter is lowered. Based on this observation, we develop an adaptive scheme to adjust this parameter during evolution, and this results in a flexible means of balancing exploration and exploitation of the search process. A comprehensive set of experiments on high dimensional benchmark functions demonstrate the usefulness of our approach.

5.1 Introduction

We start by computing the ensemble-covariance of the new population in step (8) of algorithm 12 described in chapter 2 conditional on fixing the random projection matrices $R_i, i = 1 : \tilde{M}$. We will then condition on the fit individuals and look at the effect of $R_i, i = 1 : \tilde{M}$ by computing the expectation of this ensemble covariance with respect to $R_i, i = 1 : \tilde{M}$.

Proposition 1: Conditionally on all R_i, $i = 1...\tilde{M}$, the new generation produced at Step 8 of Algorithm 12 is i.i.d. Gaussian with mean μ and the following $d \times d$ covariance matrix:

$$\Sigma_{rp} = \frac{d}{k\tilde{M}} \sum_{i=1}^{\tilde{M}} R_i^T R_i \Sigma R_i^T R_i$$

where Σ is the sample covariance of the original selected individuals in P^{Sel}

Proof: Recall from step 7(a) of Algorithm 12 that the set of projected points in the i-th subspace is:

$$\mathbf{Y}^{R_i} = \{R_i(x_1 - \mu), R_i(x_2 - \mu), ..., R_i(x_{\tilde{N}} - \mu)\}$$

Conditional on R_i, the sample covariance matrix of this set of points is:

$$\Sigma^{R_i} = \frac{1}{\tilde{N}} \sum_{n=1}^{\tilde{N}} R_i(x_n - \mu)(R_i(x_n - \mu))^T = R_i \Sigma R_i^T$$

So the samples in step 7(c) of algorithm 12 are $y_1^{R_i}, ..., y_N^{R_i} \sim N(0, \Sigma^{R_i})$.

To find the distribution of the individuals in P at step (8) of Algorithm 12, we look at the first individual:

$$P_1 := \sqrt{\frac{D\tilde{M}}{k}}[\frac{1}{\tilde{M}}\sum_{i=1}^{\tilde{M}} R_i^T y_1^{R_i}] + \mu. \tag{5.1}$$

Conditionally on $R_i, i = 1 : \tilde{M}$, this is a linear combination of independent Gaussian random variables , which is again a Gaussian[1] [62]. Hence, P_1 is Gaussian distributed, it has mean μ (since $y_1^{R_i}$ has zero mean), and we compute its covariance below.

In equation (5.1), denote $A := \sqrt{\frac{D\tilde{M}}{k}}\frac{1}{\tilde{M}}R_i^T$, then from step 7(c), we have that

$y_1^{R_i} \sim N(0, R_i \Sigma R_i^T)$.

So,

$$Ay_1^{R_i} \sim N(0, AR_i\Sigma R_i^T A^T) \tag{5.2}$$

Replacing A in (5.2), we have

$$Ay_1^{R_i} \sim N(0, \sqrt{\frac{D\tilde{M}}{k}}\frac{1}{\tilde{M}}R_i^T R_i \Sigma R_i^T \sqrt{\frac{D\tilde{M}}{k}}\frac{1}{\tilde{M}}R_i),$$

which simplifies to

$$N(0, \frac{D}{k\tilde{M}}R_i^T R_i \Sigma R_i^T R_i)$$

[1]Assume $x \sim N(m_x, \Sigma_x)$ and $y \sim N(m_y, \Sigma_y)$, then

$$Ax + By + c \sim N(Am_x + Bm_y + c, A\Sigma_x A^T + B\Sigma_y B^T)$$

.

Repeating this reasoning for each P_j, we find that:

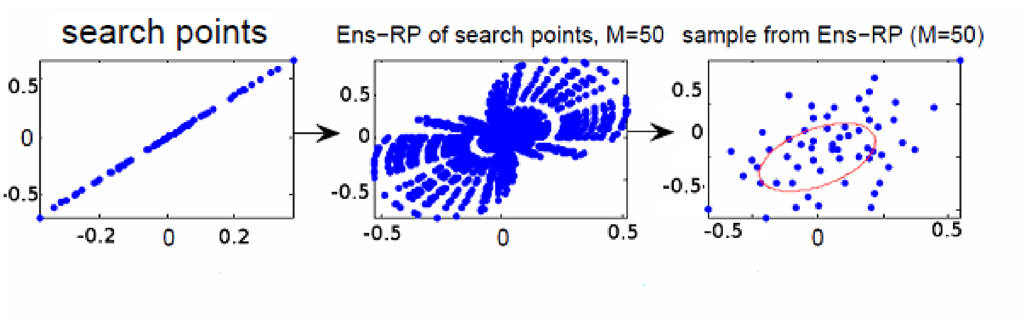
$$P_j \sim N(\mu, \frac{D}{k\tilde{M}} \sum_{i=1}^{\tilde{M}} R_i^T R_i \Sigma R_i^T R_i), \;\; j = 1, ..., N \tag{5.3}$$

Hence, the form of the ensemble covariance in the d-dimensional search space is:

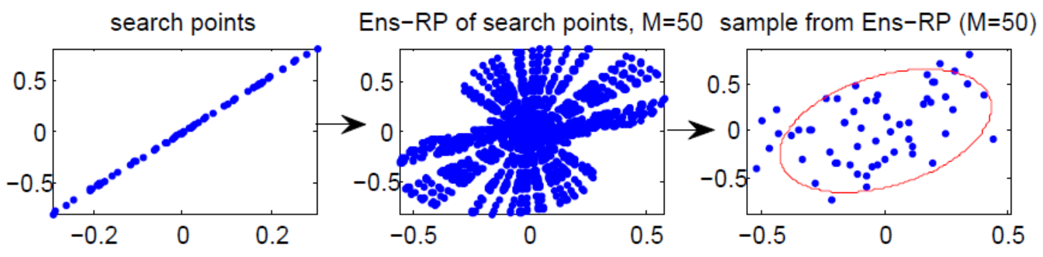
$$\Sigma_{rp} = \frac{D}{k\tilde{M}} \sum_{i=1}^{\tilde{M}} R_i^T R_i \Sigma R_i^T R_i \tag{5.4}$$

$\square$

In this chapter, Instead of generating $\tilde{M}$ independent random matrices $R_i, i = 1, ..., \tilde{M}$ with entries i.i.d from Gaussian in step (6) of algorithm 12, we generate $\tilde{M}$ independent random matrices $R_i, i = 1, ..., \tilde{M}$ with entries i.i.d from a t distribution with mean 0 and variance $\frac{1}{D}$. In this way, our expression of equation 5.4 of the RP-ENS-EDA will contain excess kurtosis of the entries of R and since the entries of our R matrices are from t distribution whose excess kurtosis contain degree of freedom, we adapt this degree of freedom in our method. This increases exploration while maintaining exploitation in our tRP-ENS-EDA whose Pseudocode is shown is algorithm 14 below. Figure 5.1 illustrates this fact. In Figure 5.1, we compare the samples from the ensemble of random projection $\tilde{M}$, whose entries are generated from Gaussian, figure 5.1(a) and the samples from the ensemble of random projection $\tilde{M}$, whose entries are generated from a heavy tail distribution (t distribution with a degree of freedom 7), figure 5.1(b). You can see the samples from the heavy tailed distribution, figure 5.1(b), more spread out with a big covariance than the sample from Gaussian distribution, figure 5.1(a), which has a smaller covariance. The bigger covariance enables the algorithm (tRP-Ens-EDA) to explore the search space better, thus escaping from a premature convergence than the smaller covariance that the algorithm (RP-Ens-EDA) has which is prone to premature convergence.



(a) Entries of R from Gaussian



(b) Entries of R from t distribution with df=7

Figure 5.1: Plots to compare covariances of the new generations of RP-Ens-EDA and tRP-Ens-EDA.

5.2 Using heavy tails in Random Projection based continuous EDA

In this section we devise a new variant of the RP-based large scale multivariate Gaussian EDA of [47] by proposing to use random matrices with heavy tailed entries. To readers familiar with the area of random projections, this might come as a surprise since random projection theory requires sub-Gaussian matrices, but our reasons will become clear in the analysis subsection shortly. In essence, as we shall see, the use of i.i.d. t-distributions, specifically its degree of freedom parameter, will allow us to enlarge the high dimensional ensemble-covariance matrix of the search distribution, which may facilitate exploration and escape early convergence, while still maintaining the focus of the search. Our analysis implies also that sub-Gaussian random matrices in this context would cause the ensemble covariance to shrink, thus making the algorithm more prone to pre-mature convergence. See figure 5.1 for the effect of the entries of R matrices on the ensemble-covariance matrix of the search distribution.

Algorithm presentation

We build on random projection ensemble based EDA (RP-Ens-EDA) [46], and refer to our new variant as tRP-Ens-EDA. The pseudocode of tRP-Ens-EDA is shown in algorithm 14.

Algorithm 14 Algorithm with entries of R from t-distribution (tRP-Ens-EDA)

```
Inputs: $k, \tilde{M}, N, MaxFE$
(1) Set $g \leftarrow 0$.
(2) Set $P \leftarrow$ Generate $N$ points randomly to give an initial population.
Do
    (3) Evaluate fitness for all $N$ points in $P$
    (4) Select the fittest $\tilde{N}$ individuals $P^{sel}$ from $P$
    (5) Estimate $\mu := mean(P^{sel})$
    (6) Generate $\tilde{M}$ independent random matrices $R_i, i = 1, ..., \tilde{M}$ with entries iid from
a t distribution with mean 0 and variance $\frac{1}{D}$.
    (7) For $i = 1, ..., \tilde{M}$.
        (a) Project the centred points into k-dimensions:
            $\mathbf{Y}^{R_i} := [R_i(x_n - \mu); n = 1, ..., \tilde{N}]$.
        (b) Estimate the $k$ x $k$ sample covariance $\Sigma^{R_i}$.
        (c) Sample $N$ new points $y_1^{R_i}, ..., y_N^{R_i} \sim_{i.i.d} N(0, \Sigma^{R_i})$.
    EndFor
    (8) Let the new population $P^{new} := \sqrt{\frac{D\tilde{M}}{k}}[\frac{1}{\tilde{M}}\Sigma_{i=1}^{\tilde{M}} R_i^T y_1^{R_i}, ..., \frac{1}{\tilde{M}}\Sigma_{i=1}^{\tilde{M}} R_i^T y_N^{R_i}] + \mu$.

    (9) $P \leftarrow P^{new}$

    (10) $g \leftarrow g + 1$
Until Termination criteria are met or MaxFE exceeded
Output $P$
```

tRP-Ens-EDA proceeds by initially generating a population of individuals randomly everywhere and selects the $\tilde{N}$ fittest points based on their fitness values. This is the set

P^{sel} in algorithm 14. The number of subspaces is denoted by $\tilde{M}$, which is a parameter. These subspaces are created in order to project the fittest individuals down to these subspaces with dimensionality $k \ll D$, where D is the dimension of the search space, and k is also a parameter of the method. For both of these parameters we will use the default values as in [46]. The other input parameters are the population size N and the maximum fitness evaluations allowed, MaxFE. Once the P^{sel} is determined, its mean is estimated in step (5) to be used in centering the points. Since we are going to have $\tilde{M}$ subspaces, $\tilde{M}$ independent random projection matrices are generated in step (6) so as to project the fittest individuals down to k dimensions in these $\tilde{M}$ subspaces. The entries of the RP matrices are drawn iid from a t distribution with mean 0 and variance $\frac{1}{D}$. This is done by sampling from $t(0, 1, \nu)$ standard t, and then multiply the samples by $\sqrt{\frac{\nu-2}{\nu D}}$. The reason we take the variance to be $\frac{1}{D}$ is to make sure we recover the original scale in Step (8) without having to modify the scaling factor: When D is high, R_i have nearly orthonormal rows if the entries have variance $1/D$. So, pre-multiplying with R_i is like orthogonally projecting the points from the D dimensional space to a k dimensional subspace, which shortens the lengths of vectors by a factor of $\sqrt{\frac{k}{D}}$ and the standard deviation gets reduced by a factor of $\sqrt{\tilde{M}}$ after averaging [46]. Therefore, the scaling factor needed to ensure this recovery is $\sqrt{\frac{D\tilde{M}}{k}}$.

Step 7(a) projects the good samples down to the subspaces of dimension k, then estimates the $k \times k$ covariance matrices for each of the subspaces and samples N new points in each subspace using the multivariate Gaussian distribution. Step (8) averages the individuals obtained from the different subspaces to produce the new population P. We should note that when D is high, any two rows of the random project matrices become quasi-orthogonal to each other [27]. There is no lower bound on k, but there are on the number of random projections, but the number of random projections has to be larger than D/k.

Analysis

Now we want to see the effect of the random R_i's on Σ_{rp}. To this end, we condition on Σ, and look at the expectation $E_R[\Sigma_{rp}]$. For this we use the following result:

Lemma 3. (Kaban, 2014 [44]): Let R be a $k \times D$ random matrix, $k < D$, with entries drawn i.i.d. from a symmetric distribution with 0-mean and finite first four moments. Let Σ be a $D \times D$ fixed positive semi-definite matrix with eigenvalues $\lambda_1, ..., \lambda_D$. Then,

$$E[R^T R \Sigma R^T R] = k \cdot E[R_{i,j}^2]^2 \Big[(k+1)\Sigma + Tr(\Sigma) I_D + \Big(\frac{E[R_{i,j}^4]}{E[R_{i,j}^2]^2} - 3\Big) \sum_{i=1}^{D} \lambda_i A_i\Big] \qquad \square$$

where A_i are $D \times D$ diagonal matrices with their jth diagonal elements being $\Sigma_{a=1}^{D} U_{ai}^2 U_{aj}^2$ and $U_{ai}$is the a-th entry of the ith eigenvector of Σ.

From (5.4), the expectation of Σ_{rp}, is

$$E[\frac{D}{k} R^T R \Sigma R^T R] = \frac{D}{k} E[R^T R \Sigma R^T R].$$

which we will compute. Since we have the entries of our t-distribution with mean 0 and variance $\frac{1}{D}$, then we will have $E[R_{ij}^2] = \frac{1}{D}$. Furthermore, we see the excess kurtosis of the entries of R featuring in this result. So we need to compute this for the t-distribution. This excess kurtosis will contain the degree of freedom parameter of the t distribution which we shall vary adaptively to control the exploration and exploitation of our algorithm.

Definition: The excess kurtosis of a random variable x is defined as:

$$K = \frac{E[x^4]}{E[x^2]^2} - 3 \tag{5.5}$$

Proposition 2: The excess kurtosis of a standardised $t(0, 1, \nu)$ distribution with degree of freedom ν, is:

$$K = \frac{6}{\nu - 4}, \quad \nu > 4 \tag{5.6}$$

To prove equation 5.6, we proceed by deriving the Generic form of the even moment, $E[x^{2k}]$, where $k = 1, ..., n$ and then instantiate it to the fourth moment, $E[x^4]$ and second moment, $E[x^2]$ and then square the second moment and plug the results in 5.5 and simplify it.

Proof: Let $x \in R^n$ be a r.v, $x \sim t(0, 1, \nu)$ and $k \in \{1, ..., n\}$, then by definition for even moments, we have

$$E[x^{2k}] = \int_{-\infty}^{\infty} x^{2k} f(x) dx$$

where $f(x)$ is the pdf of the t distribution. The pdf of a t distribution is written as:

$$f(x) = c \cdot \left(1 + \frac{x^2}{\nu}\right)^{\frac{-1}{2}(\nu+1)} dx$$

where

$$c = \frac{1}{\sqrt{\nu} B(\frac{\nu}{2}, \frac{1}{2})}$$

and B is the beta function [2]. It can be observed that $f(x) = f(-x)$, since t is a symmetric distribution. We can re-write $E[x^{2k}]$ as:

$$= \int_{-\infty}^{0} x^{2k} f(x) dx + \int_{0}^{\infty} x^{2k} f(x) dx$$

Now applying a change of variable in the first integral by letting $s = -x$, then $dx = -ds$, we will have

$$E[x^{2k}] = -\int_{\infty}^{0} (s)^{2k} f(-s) ds + \int_{0}^{\infty} x^{2k} f(x) dx$$

Interchanging bounds of the integral gives us the following

$$= \int_{0}^{\infty} (s)^{2k} f(-s) ds + \int_{0}^{\infty} x^{2k} f(x) dx$$

Since $f(-s) = f(s)$, by the property of symmetric distribution, we have

$$= \int_0^\infty (s)^{2k} f(s) ds + \int_0^\infty x^{2k} f(x) dx$$

Since $f(t) = f(x)$ and $s^{2k} = x^{2k}$, then by throwing common factor, we have

$$E[x^{2k}] = 2 \int_0^\infty x^{2k} f(x) dx \tag{5.7}$$

let $L(x) = \int_0^\infty x^{2k} f(x) dx$

Now considering, $L(x)$ and letting
$f(x) = c(1 + \frac{x^2}{\nu})^{\frac{-1}{2}(\nu+1)} dx$ where $c = \frac{1}{\sqrt{\nu} B\left(\frac{\nu}{2}, \frac{1}{2}\right)}$ and
$B\left(\frac{\nu}{2}, \frac{1}{2}\right) = \frac{\Gamma(\frac{\nu}{2})\Gamma(\frac{1}{2})}{\Gamma(\frac{\nu+1}{2})}$[2], we have

$$L(x) = c \int_0^\infty (x)^{2k} (1 + \frac{x^2}{\nu})^{\frac{-1}{2}(\nu+1)} dx$$

By change of variable, we have: $t = \frac{x^2}{\nu}$, $x = (\nu t)^{\frac{1}{2}}$, and

$$L(x) = c \int_0^\infty (\nu t)^k (1 + t)^{\frac{-1}{2}(\nu+1)} \frac{\sqrt{\nu}}{2\sqrt{t}} dt$$

$$= \frac{c}{2} \nu^{\frac{2k+1}{2}} \int_0^\infty (t)^{k-\frac{1}{2}} (1 + t)^{\frac{-1}{2}(\nu+1)} dt$$

$$= \frac{c}{2} \nu^{\frac{2k+1}{2}} \int_0^\infty (t)^{\frac{2k+1}{2}-1} (1 + t)^{-(\frac{2k+1}{2})-(\frac{\nu-2k}{2})} dt$$

This integral represents a beta function. Therefore

$$= \frac{c}{2}\nu^{\frac{2k+1}{2}} B\left(\frac{2k+1}{2}, \frac{\nu-2k}{2}\right)$$

Now plugging in the value of c back, we have

$$L(x) = \frac{1}{2\sqrt{\nu}B\left(\frac{\nu}{2},\frac{1}{2}\right)}\nu^{\frac{2k+1}{2}} B\left(\frac{2k+1}{2}, \frac{\nu-2k}{2}\right)$$

$$= \frac{1}{2}\nu^k[\Gamma(\frac{\nu}{2})\Gamma(\frac{1}{2})/\Gamma(\frac{\nu+1}{2})]^{-1}\Gamma(\frac{2k+1}{2})\Gamma(\frac{\nu-2k}{2})/\Gamma(\frac{\nu+1}{2})$$

$$= \frac{1}{2}\nu^k\frac{\Gamma(\frac{2k+1}{2})\Gamma(\frac{\nu-2k}{2})}{\Gamma(\frac{\nu}{2})\Gamma(\frac{1}{2})}, \quad \frac{\nu-2k}{2} > 0 \ \ since \ \ \Gamma(0) = \infty.$$

Now from equation 5.7, replacing $\int_0^\infty x^{2k}f(x)dx$, we have

$$E[x^{2k}] = \nu^k\frac{\Gamma(\frac{2k+1}{2})\Gamma(\frac{\nu-2k}{2})}{\Gamma(\frac{\nu}{2})\Gamma(\frac{1}{2})}, \ \ where \ \ k = 1, ...n \tag{5.8}$$

Equation (5.8) is the General form of the even moments, since we are interested in the even moments. Now instantiating (5.8) to the second and fourth moments we have

$$E[x^{2k}] = \nu^k\frac{\Gamma(\frac{2k+1}{2})\Gamma(\frac{\nu-2k}{2})}{\Gamma(\frac{\nu}{2})\Gamma(\frac{1}{2})}, \ \ where \ \ k = [1, 2] \tag{5.9}$$

Equation 5.9 is the general expression of the second and fourth moments.

Now computing the fourth moment, we let $k = 2$ in equation (5.8) and get

$$E[x^4] = \nu^2\frac{\Gamma(\frac{5}{2})\Gamma(\frac{\nu-4}{2})}{\Gamma(\frac{\nu}{2})\Gamma(\frac{1}{2})} \tag{5.10}$$

But $\Gamma(\frac{5}{2}) = \frac{3}{4}\sqrt{\pi}$ [2] and $\Gamma(\frac{1}{2}) = \sqrt{\pi}$ [2],

$$Equation\ \ 5.10 = \frac{\frac{3\nu^2}{4}\ \Gamma(\frac{\nu-4}{2})}{\Gamma(\frac{\nu}{2})}$$

By the definition of Gamma $\Gamma(x) = (x-1)\Gamma(x-1)$ [2], we have

$$E[x^4] = \frac{\frac{3\nu^2}{4}\Gamma(\frac{\nu-4}{2})}{\frac{\nu-2}{2}\frac{\nu-4}{2}\Gamma(\frac{\nu-4}{2})} = \frac{3\nu^2}{4}\frac{2}{\nu-2}\frac{2}{\nu-4}$$

$$E[x^4] = \frac{3\nu^2}{(\nu-2)(\nu-4)} \tag{5.11}$$

Analogously, computing the second moment, $E[x^2]$, we let $k = 1$ in equation (5.8) and arrive at

$$E[x^2] = \nu\frac{\Gamma(\frac{3}{2})\Gamma(\frac{\nu-2}{2})}{\Gamma(\frac{\nu}{2})\Gamma(\frac{1}{2})}$$

but again, $\Gamma(\frac{3}{2}) = \frac{1}{2}\sqrt{\pi}$ [2] and $\Gamma(\frac{1}{2}) = \sqrt{\pi}$ [2]

$$E[x^2] = \nu\frac{\frac{1}{2}\sqrt{\pi}\ \Gamma(\frac{\nu-2}{2})}{\Gamma(\frac{\nu}{2})\sqrt{\pi}} = \frac{\frac{\nu}{2}\Gamma(\frac{\nu-2}{2})}{\Gamma(\frac{\nu}{2})}$$

$$= \frac{\frac{\nu}{2}\Gamma(\frac{\nu-2}{2})}{\frac{\nu-2}{2}\Gamma(\frac{\nu-2}{2})} = \frac{\frac{\nu}{2}}{\frac{\nu-2}{2}}$$

Hence

$$E[x^2] = \frac{\nu}{\nu-2}$$

and squaring the second moment will give us

$$(E[x^2])^2 = \frac{\nu^2}{(\nu-2)^2}$$

Therefore, the excess kurtosis is

$$K = \frac{E[x^4]}{(E[x^2])^2} - 3 = \frac{\frac{3\nu^2}{(\nu-2)(\nu-4)}}{\frac{\nu^2}{(\nu-2)^2}} - 3$$

$$K = \frac{3(\nu-2)}{\nu-4} - 3 = \frac{6}{\nu-4}$$

The excess kurtosis of a t distribution of zero mean, unit variance and degree of freedom, ν is

$$K = \frac{6}{\nu-4}, \quad \nu > 4 \tag{5.12}$$

Thus the proof is done. □

Corollary: The excess kurtosis of a distribution with variance σ^2 remains unchanged.

Proof: . Let $c > 0$ be a constant. Then $c \cdot x$ has variance $c^2 var(x)$. Now show that the excess kurtosis of $c \cdot x$ is still

$$K = \frac{6}{\nu-4}, \quad \nu > 4$$

If $c > 0$, then the excess kurtosis of $c \cdot x$ is

$$\frac{E\left[(c \cdot x)^4\right]}{E\left[(c \cdot x)^2\right]^2} - 3$$

taking the constant out, we have

$$\frac{c^4 E\left[x^4\right]}{c^4 E\left[x^2\right]^2} - 3 = \frac{E\left[x^4\right]}{E\left[x^2\right]^2} - 3$$

Now the c^4 cancel out and we will be left with

$$\frac{E\left[x^4\right]}{E\left[x^2\right]^2} - 3$$

Therefore, the results is the same as the results in equation 5.5, thus the excess kurtosis did not change. $\square$

So, replacing this into Lemma 1, and noting that we can simplify the last term using $\sum_{i=1}^{D} \lambda_i A_i \preceq Tr(\Sigma) \cdot I_D$, we obtain the result:

$$\frac{D}{k} E[R^T R \Sigma R^T R] \preceq \frac{1}{D}\left[(k+1)\Sigma + Tr(\Sigma)\left(1 + \frac{6}{\nu - 4}\right) I_D\right] \tag{5.13}$$

Now let us point out what we have gained. In previous works such as [46], Gaussian was used and Gaussian corresponds to the limit when $\nu \to \infty$; therefore the expression $\frac{6}{\nu-4}$ in eq. (5.13) will tends to zero when we expand the bracket and it will tend to $Tr(\Sigma)$ if we consider the bracket as one term. so by our choice of degree of freedom that the t-distribution has, we will be adding more regularity to the covariance which also makes it larger and gives it more chance to explore the search space better. Existence of the matrix expectation we just computed requires that ν is at least 5.

Here we bring light about the statement we made at the beginning of this chapter about how an ensemble of Random Projections helps us to get a covariance close to spherical, yet this is a full covariance which can be used to interpret the correlation structure of the problem.

5.3 Setting and adaptation of the degree of freedom ν

Parameter setting methods are dichotomised into tuning and controlling [29]. Tuning means finding a good value via trial and error before running the algorithm and then fixing this value throughout the evolutionary process. On the other hand, parameter control starts with an initial value which is changed during the run, based on the feedback from the algorithm [29]. So the latter tries to adapt the control parameters automatically to adjust the algorithm to the problem while solving it during the search [84].

In our algorithm, the parameter we try to control is the degree of freedom of the t distributed entries of our random projection matrices ($R_i, i = 1 : \tilde{M}$). We have observed from our experiments on parameter tuning that different degree of freedom performed well on different problems and no single value was able to perform best on all of the problems. See figure 5.2. The comparison is done on all the functions in [64] test suite. It is hard to choose a degree of freedom that can give good results on all the problems. Because of this and the issues related to parameter tuning, we decided to control the degree of freedom adaptively. The results of our adaptive method are shown superimposed on figure 5.2 with black dashed line, and we can see that it has out performed the fixed degrees of freedom in most of the functions. Among the plots in figure 5.2 is also the results of a method which used Gaussian as entries of the random projection matrices called Gaussian RP EDA. This corresponds to the degree of freedom tending to infinity. We drew our inspiration from [84] and carved out our own adaptive method with our own rules to vary the value of the degree of freedom automatically to fit our problem. The Pseudocode of our proposed adaptive method is shown in algorithms 15, 16a, and 16b.

Algorithm 15 takes degree of freedom *df* which is an array of different values of the degree of freedom to be tried as inputs. It runs these values concurrently during a gen-

Algorithm 15 Adaptive degree of freedom ν Algorithm

```
**Inputs**: $df$: array of df values tried, $L = lenght(df)$ .
(1) for $i = 1 : L$
(2)     $dftry := df[i]$;
(3)      run steps (6),(7),(8) and (3) of Algorithm 14;
(4)     f[i] := min fitness from step (3) of algorithm 14;
(5) endfor
(6) $[fminfminInd] := min(f)$;
(7) $dfbest := df[fminInd]$;
(8) if $min(f) == max(f)$
(9)         $df[1] := round(df[1]/2)$;
(10)          for $i = 2 : L$
(11)          $df[i] == df[i] * 2$;
(12)          endfor
(13) elseif
(14)   UPDATEDF(2df) or UPDATEDF(5df);
(15) endif
Output $df$
```

Algorithm 16a UPDATEDF(2df)

```
(1) procedure UPDATEDF(2df)
(2) if $f[1] == min(f)$
(3)      $df1 := round(df1/2)$;
(4)      $df2 := round((df1 + df2)/2)$;
(5) elseif $f[2] == min(f)$
(6)      $df1 := round((df1 + df2)/2)$;
(7)      $df2 := df2 + round(df2/2)$;
(8) endif
(9) endprocedure
```

Algorithm 16b UPDATEDF(5df)

```
(1) procedure UPDATEDF(5df)
(2) if $f[1] == min(f)$
(3)     $df1 := round(df1/2);$
(4)     $df2 := round((df1 + df2)/2);$
(5)     $df3 := round((df2 + df3)/2);$
(6)     $df4 := round((df3 + df4)/2);$
(7)     $df5 := round((df4 + df5)/2);$
(8) elseif $f[2] == min(f)$
(9)     $df1 := round((df1 + df2)/2);$
(10)     $df2 := df2 + round(df2/2);$
(11)     $df3 := round((df2 + df3)/2);$
(12)     $df4 := round((df3 + df4)/2);$
(13)     $df5 := round((df4 + df5)/2);$
(14) elseif $f[3] == min(f)$
(15)     $df1 := round((df1 + df2)/2);$
(16)     $df2 := round((df2 + df3)/2);$
(17)     $df3 := df3 + round(df3/2);$
(18)     $df4 := round((df3 + df4)/2);$
(19)     $df5 := round((df4 + df5)/2);$
(20) elseif $f[4] == min(f)$
(21)     $df1 := round((df1 + df2)/2);$
(22)     $df2 := round((df2 + df3)/2);$
(23)     $df3 := round((df3 + df4)/2);$
(24)     $df4 := df4 + round(df4/2);$
(25)     $df5 := round((df4 + df5)/2);$
(26) elseif $f[5] == min(f)$
(27)     $df1 := round((df1 + df2)/2);$
(28)     $df2 := round((df2 + df3)/2);$
(29)     $df3 := round((df3 + df4)/2);$
(30)     $df4 := round((df4 + df5)/2);$
(31)     $df5 := df5 + round(df5/2);$
(32) endif
(33) endprocedure
```

eration and keeps track of the best fitnesses found with each of these parameter values. These fitnesses of each of the df tried are compared to see which one is the best and then choose the degree of freedom that gives the best fitness to be taken forward. The contents of df are then updated accordingly, in a way that places the values that are to be tried next time around the best performing one. If all values tried performed the best then we spread out the content of df. The rules of how this is done are given in algorithms 16a and 16b, where we use 2 or 5 different values concurrently respectively. This process is then repeated in each generation. For instance, in the case of the method that uses 2 dfs concurrently, say 2 and 4. We run steps (6), (7), (8) and (3) of algorithm 15 with $df_1 = 2$ and $df_2 = 4$ concurrently and compare the fitness values obtained with these two dfs from step (3). If the fitness value obtained by say, $df_1 = 2$ is better than the fitness value obtained by say $df_2 = 4$, we complete the generation with the $df_1 = 2$ and update the df_1 and df_2 to be used in the next generation as following: $df_1 := round(df_1/2)$ and $df_2 := round((df_1 + df_2)/2)$, according to algorithm 16a.

The number of values for the degree of freedom parameter that are tried concurrently at each generation need not be exactly two or five, and the rules can easily be designed for different numbers if desired. The more degree of freedoms used concurrently, the better the results, but this comes at the expense of more function evaluations.

5.4 Analysis of the experimental results

This section is devoted to testing our idea of heavy tailed entries in the random matrices, and compare the performance of our new approach with the known method of using Gaussian entries to see if our proposal is superior. Finally we put the results in context by comparing our method with existing state of the art.

Table 5.1: 1000-dimensional test functions from the CEC'10 collection.

Problem	Name	Type
F2	Shifted Rastrigin's function	Multimodal
F3	Shifted Ackley's function	Multimodal
F5	Single-group Shifted and m-rotated Rastrigin's function	Multimodal
F6	Single-group Shifted and m-rotated Ackley's function	Multimodal
F10	$\frac{D}{2m}$-group Shifted and m-rotated Rastrigin's function	Multimodal
F11	$\frac{D}{2m}$-group Shifted and m-rotated Ackley's function	Multimodal
F13	$\frac{D}{2m}$-group Shifted and m-dimensional Rosenbrock's function	Multimodal
F15	$\frac{D}{2m}$-group Shifted and m-rotated Rastrigin's function	Multimodal
F16	$\frac{D}{m}$-group Shifted and m-rotated Ackley's function	Multimodal
F18	$\frac{D}{m}$-group Shifted and m-dimensional Rosenbrock's function	Multimodal
F20	Fully nonseparable Rosenbrock	Multimodal

5.4.1 Benchmark functions used

We use two sets of benchmark functions: the 1000-dimensional CEC'2010 test suite as described in [1], and the 50-dimensional CEC'2005 test suite [64]. All problems are minimizations. The majority of the test functions from the CEC'2010 benchmarks contain are non-separable, and hence harder to optimise.

In the CEC'2005 problems, 5 are unimodal and 11 multimodal. All the global optima are within the given box constrains. However, problem 7 is without a search range and with the global optimum outside of the specified initialization range. From the problems in the [1] suite we are most interested in the multimodal ones.

Table 5.1 lists the 1000-dimensional CEC'10 problems that we used, and Table 5.2 gives the 50-dimensional ones.

5.4.2 Experiments and Results

We conducted three types of experiments. The first is with a fixed time frame of 300 generations in order to assess the potential of various values for the degree of freedom (df) as well as our adaptive procedures. The second type of experiment compares tuning with adaptation on a fixed budget of function evaluations, set to $5.4 \cdot 10^5$ and the third

Table 5.2: 50-dimensional test functions from the CEC'05 collection.

Problem	Name	Type
P01	Shifted Sphere Function	Unimodal
P02	Shifted Schwefels Problem 1.2	Unimodal
P03	Shifted Rotated High Conditioned Elliptic Function	Unimodal
P05	Schwefel's Problem 2.6 with Global Optimum on Bounds	Unimodal
P06	Shifted Rosenbrock's Function	Multimodal
P07	Shifted Rotated Griewank's Function without Bounds	Multimodal
P08	Shifted Rotated Ackley's Function with Global Optimum on Bounds	Multimodal
P09	Shifted Rastrigin's Function	Multimodal
P10	Shifted Rotated Rastrigin's Function	Multimodal
P11	Shifted Rotated Weierstrass Function	Multimodal
P12	Schwefel's Problem 2.13	Multimodal
P13	Expanded Extended Griewank's plus Rosenbrock's Function	Multimodal
P14	Expanded Rotated Extended Scaffe's F6	Multimodal
P15	Shifted Griewank's Function	Multimodal
P16	Shifted Ackley's Function	Multimodal

Table 5.3: Ranksum statistical test for performance comparison between Tuning, 5df and 2df methods ran on equal budget.

Problems	**Tuning**	**2 concurrent dfs**	**5 concurrent dfs**
	(mean Optimal Gap)	**(mean Optimal Gap)**	**(mean Optimal Gap)**
P01	0±0	0±0	0±0
P02	4.07e+04±7.32e+03	1.53e+04±3.85e+03	1.62e+04±3.23e+03
P03	7.84e+06±1.74e+06	2.06e+06±3.82e+05	2.29e+06±4.87e+05
P04	**0±0**	2.17e+04±4.56e+03	2.05e+04±4.99e+03
P05	3.99e+03±401.26	4.27e+03±406.49	4.18e+03±268.13
P06	3.01e+03±3.14e+03	894.65±1.31e+03	1.02e+03±1.62e+03
P07	0.94±0.11	1.48e-05±1.44e-05	1.14e-05±8.43e-06
P08	21.19±0.04	**21.14±0.06**	21.18±0.04
P09	332.32±13.52	**313.40±11.45**	326.12±10.58
P10	341.78±15.66	**316.36±14.23**	326.60±11.13
P11	74.45±1.79	73.74±1.17	73.67±1.67
P12	5.61e+06±3.38e+05	**5.27e+06±3.03e+05**	5.55e+06±5.10e+05
P13	30.35±1.08	28.36±1.14	29.46±0.83
P14	23.08±0.16	**22.9686±0.15**	23.08±0.15
P15	7.79e-11±7.04e-011	0±0	0±0
P16	3.37e-13±1.05e-013	2.84E-14±0	2.84E-14±0

compares our method with other state of the arts. In all the experiments, 25 independent runs were performed on all the methods.

RP-Ens-EDA vs. the proposed tRP-Ens-EDA in experiments with equal time frame

We use the 50 dimensional CEC 2005 benchmark functions. We ran the RP-Ens-EDA [46] and our proposed tRP-ENS-EDA with different degrees of freedom values for 300 generations each. The purpose was to see the potential of changing the df online. We decided to stop at 300 generations because that is where the performances of the different methods can be noticed. Beyond that, everything remain the same. Most of them are at their maximum potential and will not improve in the remaining generations. See the results in figure 5.2. We used 5df to compare the less performing method (5df) with RP-END-EDA and once it is outperformed, we did not have to compare it with the better performing one (2df).

As we can see from the results in figure 5.2, different degrees of freedom perform well on different problems. Therefore, we also ran our proposed adaptive method to automatically select the value of the degree of freedom as the optimisation progresses to optimise each problem.

From the comparisons shown in figure 5.2, we can see that our method, tRP-Ens-EDA(5df) has demonstrated superiority over RP-Ens-EDA by outperforming it in 8 out of the 16 problems and it has almost the same performance with RP-Ens-EDA on 2 of the problems with RP-Ens-EDA outperforming our method in only two problems. Therefore, we can conclude that our method, tRP-Ens-EDA(5df) is superior to RP-Ens-EDA. However, this is not surprising since it used more budget of function evaluations per generations due to the number of concurrent trials it had to make. We also performed experiments with the version of our adaptive method that uses 2 values concurrently, and found, as expected, that it performs slightly inferior to the version that uses 5 concurrent values – although we need to bear in mind the trade-off that using more concurrent values means more fitness evaluations per generation.

Tuning vs. adaptation in equal budget experiment

In this set of experiments, we compare tuning with adaptation under equal budget of total number of function evaluations. Tuning encompasses separate runs with the degree of freedom parameter being fixed to a value in the set $\{5, 6, 7, 10, 12, 30\}$. The number of generations was 300 with a population size of 300. This means the total number of function evaluations available for the tuning method was $5.4 \cdot 10^5$. Hence we gave the same amount of function evaluations to our two adaptive methods. This means that tRP-Ens-EDA(2df) ran for 900 generations with a population size of 300 and tRP-Ens-EDA(5df) ran for 360 generations with same population size. The aggregated results are summarised in figure 5.3. We see that our method with 2 concurrent values of degree of freedom has out performed the other two. This is confirmed in a statistical test results in table 5.3 where bold font indicates statistical out-performance.

Therefore, with equal budget, it is better to use the method that uses less df, say 2df concurrently as it is more cost effective than the other methods. This is true because the tuning method only uses pre-defined dfs to choose from and does not try all other possible values, hence the inferiority. Our method that uses 5 df concurrently uses lots of function evaluations just to try out possible dfs and does not have much left to create new generations, while our version that uses 2 df concurrently uses less function evaluations than the other two methods to try out possible dfs and greater amount of function evaluations remain to be used to create new generations. This behaviour turns out to be advantageous and performs better than the rest of the methods tested.

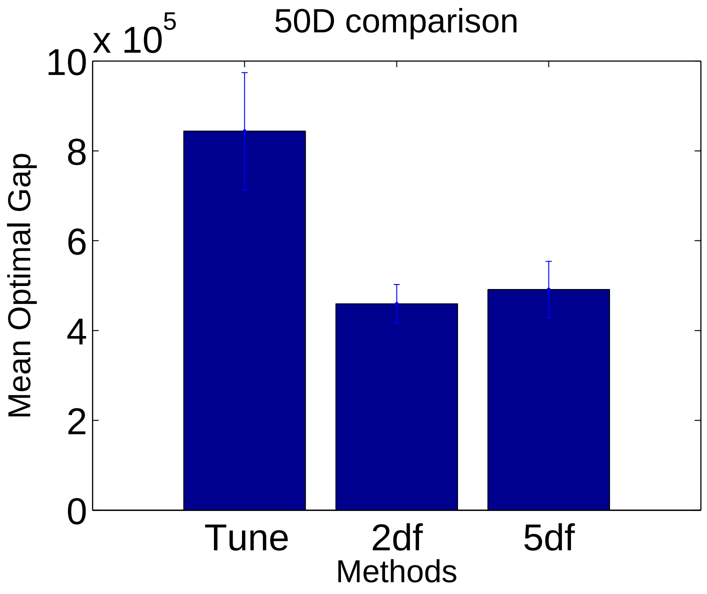


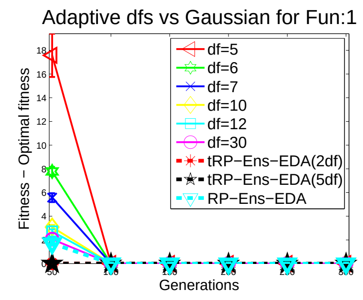
Figure 5.3: Aggregated Summary of the comparison experiment with equal budget set to $5.4 \cdot 10^5$ on 50-dimensional Rosenbrock function.

Comparison with state of the art methods on 1000 dimensional problems

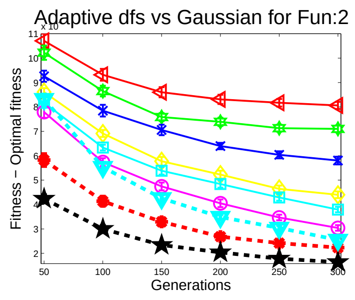
In this section, we show results of comparing our method $tRP - Ens - EDA(2df)$, with other state of the art performing methods on the 1000 dimensional CEC 2010 problems. The results of these comparisons are summarised in tables 5.4, 5.5 and 5.6. We present the average and std of best fitness results after 3 million, 1.2 million and 0.6 million function evaluations respectively. Bold font indicates statistical out-performance using Friedman Test. We present the results of our Friedman multicomparison test graphically in figure 5.4 for visualisation. Friedman test is a non-parametric statistical test, which is used for detecting the difference between many (normally more than 3/2) related samples/Data [21],[17]. After the Friedman test, follows a multicomparison test [17], which graphically shows the difference between pair of the methods. The P-value determines significance difference. If the Value is less or equal to 0.05 when 5 percent significance level is considered, then there exist at least one of the samples, which is different from the rest. If on the other hand, the P-value is greater than 0.05 at 5 percent significance level, then no differences are detected between the performances of the methods compared. The blue line in figure 5.4 represents our method which is compared with other methods. If any of the other competing methods register a red line against them, it means the difference in performance between that method and ours by Friedman test is statistically significant. Overlapping intervals indicate no significant difference. After conducting Friedman test to establish statistical significant differences, we do a multicomparison test. From tables 5.4, 5.5 and 5.6, we see our method is a par with the state of the art methods and it is not worst than them. Therefore, our method has once again proved to be very promising.

5.5 Summary

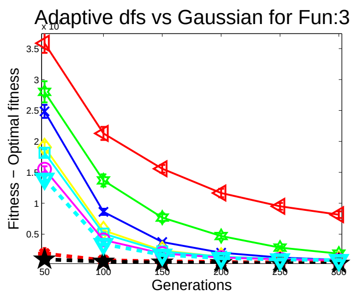
We devised a new approach for high dimensional continuous black-box optimization by building on RP ensemble based EDA. Our focus has been on the utility of the heavy tailed distributions as entries of our RP matrices. Our results suggest that taking RP matrices with entries drawn with i.i.d. t-distribution increases exploration and at the same time maintains exploitation and focus. We have demonstrated superiority of our method against the method we built on, RP-ENS-EDA. We have also demonstrated competitiveness of our method in comparison with a number of state of the art methods on 1000 dimensional problems on three different budget sizes.



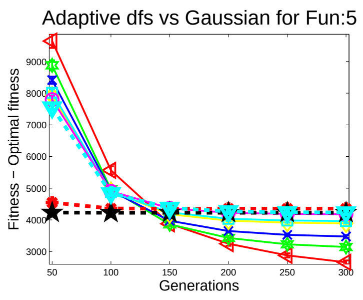
(a) Function 1



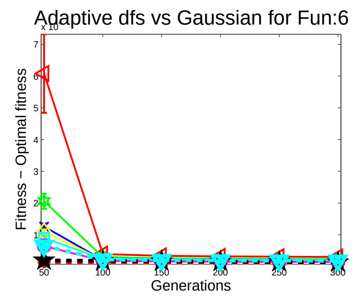
(b) Function 2



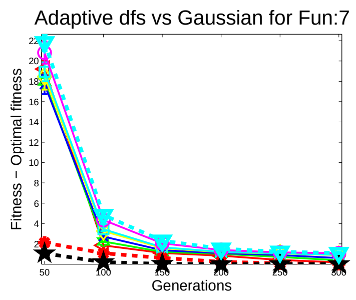
(c) Function 3



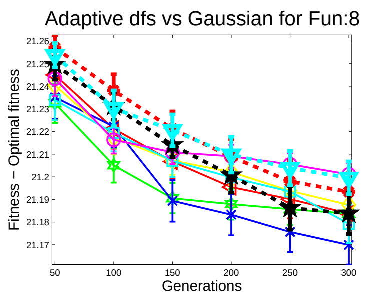
(d) Function 5



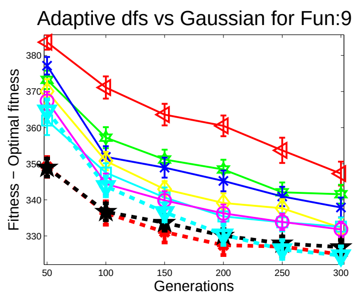
(e) Function 6



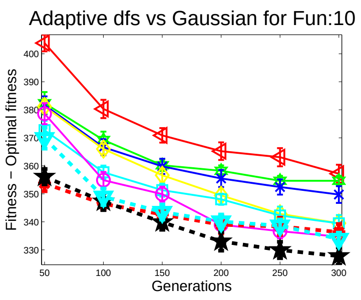
(f) Function 7



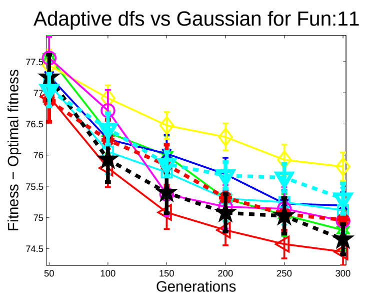
(g) Function 8



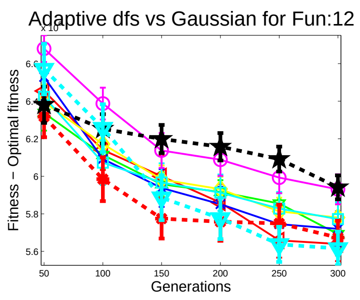
(h) Function 9



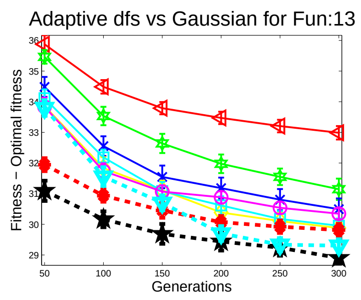
(i) Function 10



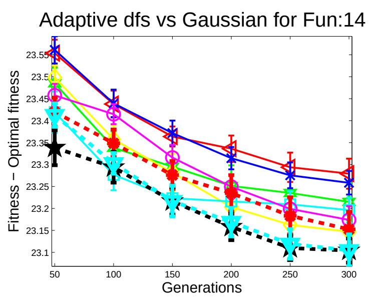
(j) Function 11



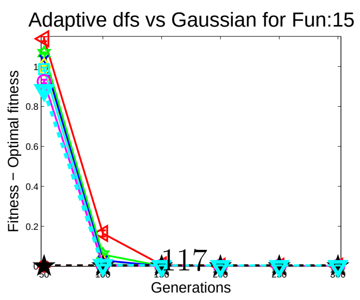
(k) Function 12



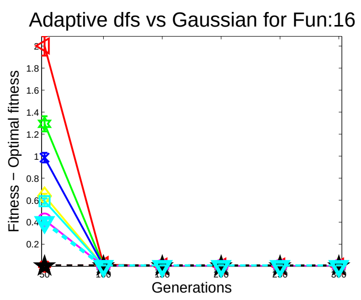
(l) Function 13



(m) Function 14



(n) Function 15



(o) Function 16

Figure 5.2: Plots to compare different dfs, RP-Ens-EDA and two versions of our tRP-Ens-EDA on Functions 1-16. For better visibility, we display from generation 50 only and show legend of only the first plot. The error bars represent one standard error over 25 repeated runs.

Table 5.4: Comparison with state of the art under equal budget of $3 \cdot 10^6$ function evaluations.

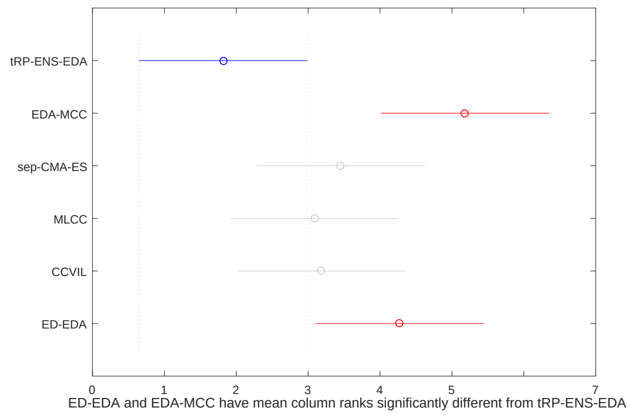
	ED-EDA		CCVIL		MLCC		sep-CMA-ES		EDA-MCC		tRP-Ens-EDA	
	Mean	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
F2	1.11E+04	1.12E+02	**4.00E-07**	**0.00E+00**	37.89	86.41	5.68E+03	4.89E+02	1.17E+04	6.38E+01	5.94E+02	1.76E+01
F3	7.22E-01	4.71E-02	4.00E-07	6.32E-07	8.45E-01	1.04E+00	2.15E+01	1.08E-01	2.92E+00	7.76E-02	**2.48E-13**	**4.57E-15**
F5	3.15E+08	1.33E+07	5.51E+08	1.55E+08	1.22E+08	8.63E+07	1.19E+08	2.92E+07	4.24E+08	1.67E+07	**4.99E+06**	**1.32E+06**
F6	3.56E+06	2.81E+05	4.14E+05	6.54E+05	1.06E+06	9.42E+05	6.39E+06	3.85E+06	1.53E+07	1.92E+05	3.20E+01	5.32E-01
F10	1.12E+04	1.00E+02	1.43E+03	6.34E+01	2.93E+03	6.72E+02	6.28E+03	2.51E+02	1.18E+04	6.35E+01	**5.96E+02**	**1.71E+01**
F11	1.49E+02	2.33E+01	**7.44E+00**	**2.41E+00**	1.64E+02	7.72E+00	2.12E+02	6.16E+00	1.91E+02	7.39E-01	3.10E+01	1.09E+01
F13	2.13E+06	2.25E+05	2.98E+11	8.57E+11	1.56E+03	5.52E+02	**2.94E+02**	**9.20E+01**	4.77E+10	2.99E+09	1.00E+06	5.74E+04
F15	1.13E+04	9.74E+01	2.78E+03	8.78E+01	7.11E+03	1.34E+03	6.76E+03	2.76E+02	1.18E+04	6.61E+01	**6.05E+02**	**2.74E+01**
F16	1.76E+02	2.35E+01	1.31E+01	2.92E+00	3.62E+02	7.80E+00	4.21E+02	1.59E+01	2.37E+02	2.43E+01	8.04E+01	1.04E+01
F18	1.02E+06	3.72E+05	6.42E+11	2.19E+12	3.36E+03	9.08E+02	**9.16E+02**	**1.94E+02**	1.49E+10	1.67E+09	4.41E+04	5.40E+03
F20	1.89E+05	9.81E+04	1.75E+11	8.77E+11	2.23E+03	3.20E+02	9.04E+02	3.91E+01	8.08E+08	7.81E+07	1.02E+03	2.83E+01

Table 5.5: Comparison with state of the art under equal budget of $1.2 \cdot 10^6$ function evaluations.

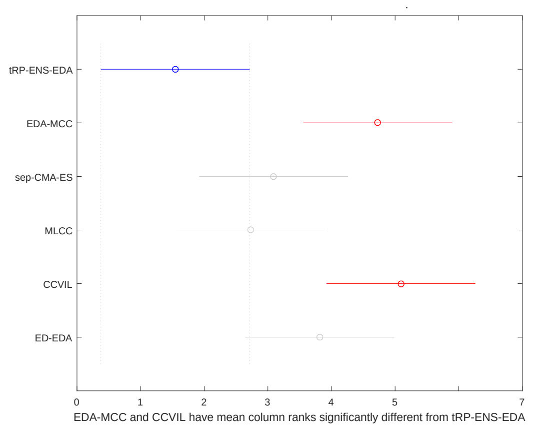
	ED-EDA		CCVIL		MLCC		sep-CMA-ES		EDA-MCC		tRP-Ens-EDA.	
	Mean	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
F2	1.23E+04	1.18E+02	**1.30E+01**	**1.00E+01**	989.99	751.00	7.24E+03	4.21E+02	1.26E+04	6.55E+01	5.79E+02	2.40E+01
F3	1.68E+01	1.15E-01	1.30E+01	9.08E+00	5.97E+00	3.79E+00	2.15E+01	1.09E-01	1.56E+01	8.17E-02	**2.47E-13**	**4.50E-13**
F5	3.57E+08	1.62E+07	9.53E+08	1.01E+08	3.04E+08	1.73E+08	1.18E+08	2.92E+07	4.56E+08	1.17E+07	1.46E+08	1.34E+08
F6	5.78E+06	2.07E+05	2.12E+07	3.41E+05	6.10E+06	6.45E+06	6.39E+06	3.84E+06	1.62E+07	3.27E+05	**2.13E+01**	**2.69E-02**
F10	1.24E+04	1.13E+02	1.32E+04	3.32E+02	4.68E+03	1.43E+03	7.43E+03	3.86E+02	1.27E+04	8.04E+01	6.01E+02	2.66E+01
F11	1.96E+02	6.19E+00	2.32E+02	6.66E-01	1.81E+02	7.06E+00	2.13E+02	6.80E+00	2.05E+02	5.77E-01	3.50E+01	9.49E+00
F13	4.53E+09	4.10E+08	8.45E+10	1.55E+11	**3.70E+04**	**5.03E+04**	5.96E+02	1.73E+02	9.51E+10	4.96E+09	1.27E+06	5.38E+04
F15	1.25E+04	1.17E+02	1.83E+04	5.65E+02	1.17E+04	7.32E+03	7.36E+03	3.44E+02	1.27E+04	6.95E+01	**5.99E+02**	**2.46E+01**
F16	3.47E+02	4.88E+00	4.26E+02	7.59E-01	3.69E+02	6.86E+00	4.31E+02	1.00E+01	3.43E+02	3.12E+00	**9.18E+01**	**1.57E+01**
F18	5.62E+10	3.50E+09	3.60E+11	3.91E+11	8.89E+05	2.90E+06	**1.45E+03**	**3.10E+02**	1.03E+11	4.48E+09	4.16E+04	9.34E+03
F20	7.07E+10	4.10E+09	2.69E+11	3.78E+11	3.00E+06	1.09E+07	1.05E+03	5.16E+01	8.99E+10	4.43E+09	1.08E+03	5.45E+01

Table 5.6: Comparison with state of the art under equal budget of $0.6 \cdot 10^6$ function evaluations.

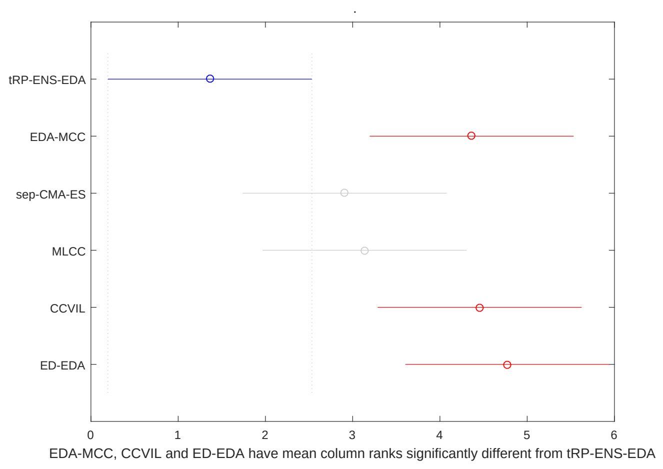
	ED-EDA		CCVIL		MLCC		sep-CMA-ES		EDA-MCC		tRP-Ens-EDA.	
	Mean	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**	**Mean**	**Std**
F2	1.59E+04	1.22E+02	**1.77E+01**	**1.40E+01**	6.22E+03	1.75E+03	8.15E+03	3.63E+02	1.51E+04	5.61E+01	5.79E+02	2.40E+01
F3	2.08E+01	3.12E-02	1.77E+01	1.97E+00	1.45E+01	1.90E+00	2.15E+01	2.07E-02	2.05E+01	2.33E-02	**2.48E-13**	**4.76E-15**
F5	4.53E+08	1.58E+07	9.79E+08	7.60E+07	4.19E+08	1.76E+08	1.41E+08	3.00E+07	5.08E+08	1.41E+07	3.04E+08	9.49E+06
F6	1.13E+07	4.79E+05	2.12E+07	3.76E+05	1.41E+07	8.29E+06	1.04E+07	2.33E+06	1.74E+07	3.01E+05	**5.48E+02**	**7.50E+01**
F10	1.61E+04	1.61E+04	1.31E+04	3.29E+02	1.10E+04	1.97E+03	8.36E+03	3.85E+02	1.52E+04	9.69E+01	6.01E+02	2.66E+01
F11	2.24E+02	7.90E-01	2.32E+02	8.08E-01	2.16E+02	1.79E+01	2.25E+02	5.05E+00	2.22E+02	5.33E-01	3.50E+01	9.49E+00
F13	2.77E+11	1.36E+10	1.84E+11	1.23E+11	1.21E+09	2.15E+09	5.02E+05	8.66E+04	2.90E+11	4.49E+09	1.40E+06	5.67E+04
F15	1.56E+04	3.08E+02	1.86E+04	3.76E+02	1.84E+04	1.02E+04	8.13E+03	3.49E+02	1.52E+04	1.93E+02	**5.99E+02**	**2.46E+01**
F16	4.14E+02	5.84E-01	4.26E+02	9.41E-01	4.14E+02	1.22E+01	4.30E+02	2.88E+00	4.08E+02	8.34E-01	**9.18E+01**	**1.57E+01**
F18	9.77E+11	3.14E+10	4.76E+11	2.86E+11	2.67E+10	5.31E+10	2.19E+04	1.12E+04	7.81E+11	8.81E+09	4.61E+04	9.72E+03
F20	1.13E+12	2.82E+10	4.48E+11	2.64E+11	5.26E+10	7.80E+10	1.16E+03	1.34E+02	8.83E+11	2.86E+10	1.10e+03	6.76E+01



(a) Under equal budget of $3 \cdot 10^6$ function evaluations



(b) Under equal budget of $1.2 \cdot 10^6$ function evaluations



(c) Under equal budget of $0.6 \cdot 10^6$ function evaluations

Figure 5.4: Friedman multicomparison statistical test of our tRP-ENS-EDA with other state of the arts methods. Overlapping intervals indicate no significant difference

CHAPTER 6

Random Embedding in Estimation of Distribution Algorithm(REMEDA)

It has been observed that in many real-world large scale problems only few variables have a major impact on the function value: While there are many inputs to the function, there are just few degrees of freedom. We refer to such functions as having a low intrinsic dimension. In this chapter we devise an Estimation of Distribution Algorithm (EDA) for continuous optimisation that exploits low intrinsic dimension without knowing the influential subspace of the input space, or its dimension, by employing the idea of random embedding. While the idea is applicable to any optimiser, EDA is known to be remarkably successful in low dimensional problems but prone to the curse of dimensionality in larger problems because its model building step requires large population sizes. Our method, Random Embedding in Estimation of Distribution Algorithm (REMEDA) remedies this weakness and is able to optimise very large dimensional problems as long as their intrinsic dimension is low.

6.1 Introduction

Optimisation over a high dimensional search space is challenging. High dimensionality limits the usefulness of problems in practice. However, there are problems that are high dimensional but can possibly be represented in low dimension [18]. This might render very high dimensional problems solvable, as they are in fact not truly high-dimensional [82]. It has also been noted that in certain classes of functions most decision variables have an insignificant impact on the objective function. Examples include hyperparameter optimization for neural and deep belief networks [9], automatic configuration of state-of-the algorithms for solving NP-hard problems [41], optimisation problems in robotics [82], and others [18]. In other words, these problems have low *intrinsic dimensionality*. In the numerical analysis literature [18], the influential parameter subspace has been termed as the ”active subspace”, and methods have been developed to estimate this subspace.

Fortunately, for optimisation, estimating the influential subspace is not required: In [82], it was shown that a sufficiently large *random* subspace contains an optimum with probability 1, and this was used to dramatically improve the efficiency of Bayesian optimisation by exploiting the low intrinsic dimensionality of problems.

In this chapter,[1] we employ the random embedding technique and develop it further to scale up Estimation of Distribution Algorithms (EDA) for problems with low intrinsic dimension. Although the underlying theoretical considerations are applicable to any optimisation method, our focus on EDA is due to it being one of the most successful methods in low dimensional problems [73] and most unsuccessful or expensive in high dimensions [22, 47, 74].

[1]A shorter version of work presented in this chapter appears in Proc. of the 14-th International Conference on Parallel Problem Solving from Nature (PPSN XIV), 17-21 September, Edinburgh, Scotland, 2016. **This work was nominated for best paper award.**

6.2 Intrinsic dimension of problems

High dimensionality limits the usefulness of problems in practice. However, it is most of the time possible to represent high dimensional problems in low dimension. This means the functions only varies along a few independent directions. This is because they are being embedded in high dimension, but they are not really high dimensional. So we can represent them in their low intrinsic dimensions. Therefore, if the methods are able to exploit low intrinsic structure then very high dimensional problems might become manageable due to the fact that they are not truly high-dimensional. Instead, the structure is being embedded in high dimensional space [82] and can be appropriately represented in a much lower dimension. The maximum number of dimensions that impacts significantly on the objective function are called the *intrinsic dimensions*, otherwise known as the *effective dimensions* of a function, which is normally lower than the ambient dimension.

Definition. A function $f : \mathbb{R}^D \to \mathbb{R}$ has **intrinsic dimension** d_i, with $d_i < D$, if there exists a d_i dimensional subspace Υ such that $\forall x \in \mathbb{R}^D$, $f(x) = f(\text{Proj}_\Upsilon(x))$.
In the above, $\text{Proj}_\Upsilon(x)$ denotes the orthogonal projection, i.e. $\text{Proj}_\Upsilon(x) = \Phi\Phi^T x$, where $\Phi \in R^{D\times d_i}$ has columns holding a linear basis of Υ.

The following result in [82] shows that, for such functions, a global optimum exists in a randomly chosen linear subspace – hence a low dimensional search is sufficient.

Theorem 6.2.1 ([82]). Assume we are given a function $f : \mathbb{R}^D \to \mathbb{R}$ with intrinsic dimension $d_i < d$ and a random matrix $R \in \mathbb{R}^{D\times d}$ with independent entries sampled from a standard Gaussian. Then, with probability 1, for any $x \in \mathbb{R}^D$, there exists a $y \in \mathbb{R}^d$ such that $f(x) = f(Ry)$. $\square$

Figure 6.1 illustrates theorem 6.2.1 above. The figure shows an 1D subspace denoted as Υ or T in the figure embedded in a 2D ambient space. The red line which is labelled as R, represents a randomly oriented subspace of dimension of 1. Recall that columns of R are from a d dimensional subspace and $d = 1$ here. Here $d_i = d = 1$ and $D = 2$. Let x^* be a global optimiser of the function. The dashed line is orthogonal to Υ labelled as T in the figure, going through x^*. Due to the special property of the functions, all points on the dashed line has the same function value as x^*. The claim is, that there exist an intersection point between the dashed line and the R. Therefore, it is enough to search on R for a point that has the same function value as x^*.

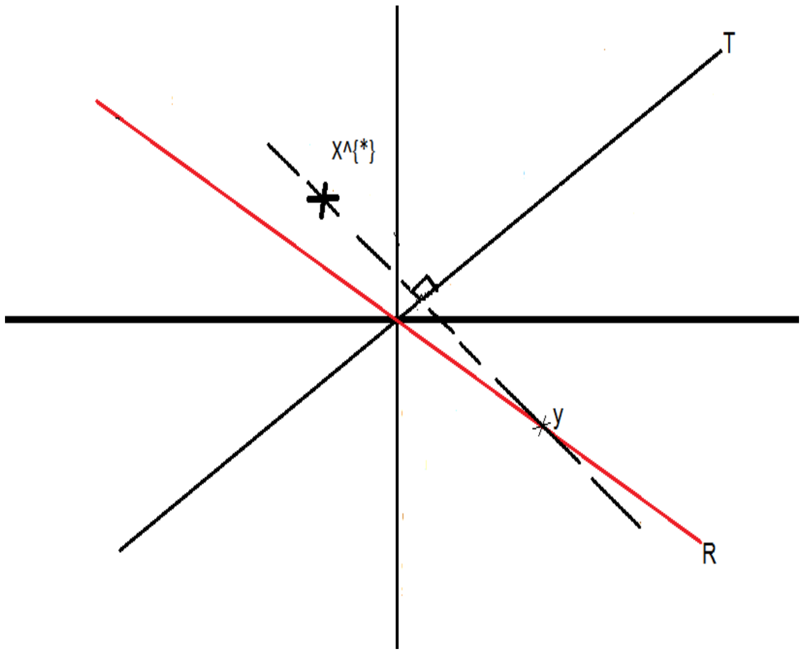


Figure 6.1: Gaurantee that the optimum is in the subspace.

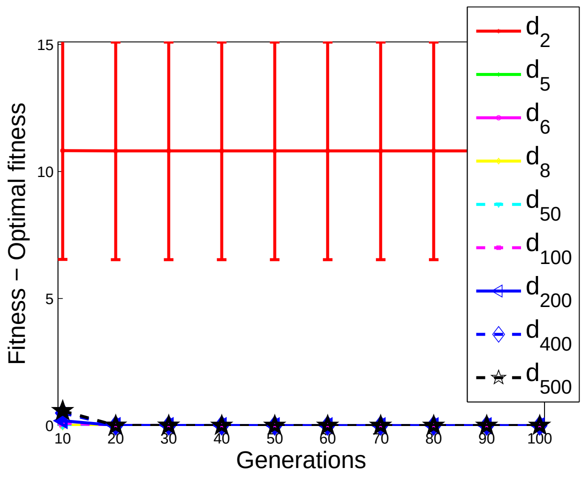
Given some box constraints on the original problem, the authors [82] develop an upper bound on the search box required for the low dimensional search. However, their proof only applies to the case when $d = d_i$, and in practice they recommend a smaller search box and use a slightly larger d. Recall, in practice we have no knowledge of the value of d_i. However, on synthetic problems, we conducted some experiments to try different values of the internal dimension, d to see which values of d will contain the optimum in the d-dimensional search box. The experimental results do appear to be better, with the optimum existing in the d-dimensional box, when d is slightly larger than d_i. See figure 6.2. The authors in [82] derived a bound on their box size, d but did not use it in practice, instead they suggested using $[-\sqrt{d}\ \sqrt{d}]$. This was the box size they used in all their experiments. We used the same box size to run our experiments, but their recommended box size does not work when $d = d_i$. Therefore, we use the box size they derived from theory and found out that it only works when $d = d_i$. See figure 6.3 for the results of the experiment we carried out using different values of $d = d_i$. We conducted experiemnts by trying different values of $d = d_i$ on the box size authors in [82] derived to check if that will work, since their recommended box size did not work when $d = d_i$. From the results of these experiments, we can see that the box authors in [82] derived is quite necessary since the algorithm works on $d = d_i$. We have also notice that in the experimental part of [82], they have $d > d_i$ and their theory does not apply in this case. Therefore, we fill this gap in the next section.

In the next section we derive a bound on the search box that holds true for $d > d_i$, and show that the required box size that guarantees to contain a global optimum is indeed smaller when d is larger. Secondly, we devise an EDA optimisation algorithm that implements these ideas employing a random Gaussian embedding. In a subsequent section we also extend this to sub-Gaussian embeddings to increase the efficiency of implementation.

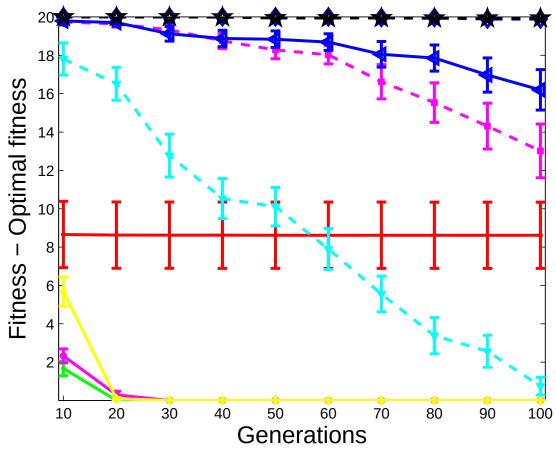
6.3 REMEDA: Random Embedding EDA

In this section we present our REMEDA algorithm and explain how it exploits the low intrinsic dimensionality of problems. Instead of optimising in the high dimensional space, REMEDA will do a random embedding, using the random matrix $R \in \mathbb{R}^{D \times d}$, $d << D$ with i.i.d. entries drawn from a standard Gaussian, and then optimises the function $g(y) = f(Ry)$, $y \in \mathbb{R}^d$ in the lower dimensional space.

The psuedo-code of REMEDA is given in algorithm 17. It takes the population size



(a) Shifted Sphere Function



(b) Shifted Rosenbrock Function

Figure 6.2: Trajectories showing different sizes of the internal dimension, d varied and $d_i = 2$. They are averaged over 25 independent runs.

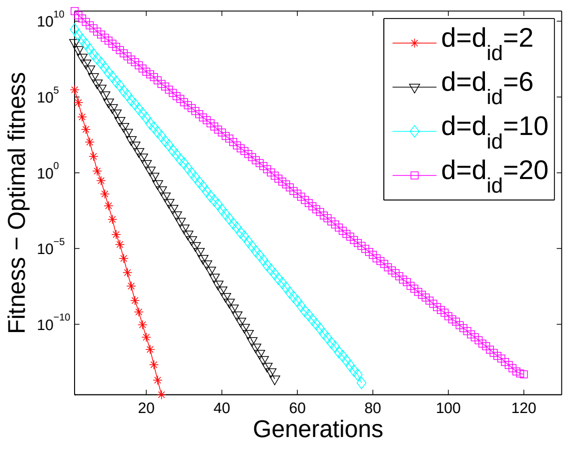


Figure 6.3: Trajectories showing results on the shifted sphere function when $d = d_i$. These results are averaged over 25 independent restarts

N, box constraints for a D-dimensional problem, and the internal working dimension $d << D$. As in basic EDA, the REMEDA algorithm then proceeds by initially generating a population of individuals uniformly randomly. However, these individuals are generated in the d-dimensional space, within some suitable box constraints in this space that are determined from the given D-dimensional box. The details of how this is done will follow shortly. The algorithm then evaluates the fitness of these individuals with the use of a random embedding matrix $R \in \mathbb{R}^{D \times d}$ that transforms the d-dimensional points into the original D-dimensional space of decision variables. The matrix R has entries drawn i.i.d. from a standard Gaussian distribution, as in Theorem 1, i.e the first theorem. Based on the fitness values obtained, the fittest individuals are selected using a selection method, such as truncation selection. The maximum likelihood estimates (MLE) of the mean $\mu \in \mathbb{R}^d$ and the covariance $\Sigma \in \mathbb{R}^{d \times d}$ of the promising solutions are computed from the set of selected fittest individuals, and these are used to generate the new generation by sampling from a multivariate Gaussian distribution. The new population is formed by replacing the old individuals by the new ones. We also use elitism, whereby the best individual of the previous generation is kept.

Algorithm 17 The Pseudocode of REMEDA with Population size N and low intrinsic dimensionality of the problems, d_i

```
**Inputs:** $N$, $D$, $d$, Box
(1) Set the search box boundaries in the low-dimensional space $\mathbb{R}^d$ (cf. Theorem 2(6.3.1) & text)
(2) Set $P \leftarrow$ Generate $N$ points uniformly randomly within the box in $\mathbb{R}^d$ to give an initial population
(3) Set $R \leftarrow$ Generate a random embedding matrix, $R \in \mathbb{R}^{D \times d}$.
**Do**
        (4) Evaluate the fitness of $y_i$ as $f(Ry_i)$, $i = 1...N$
        (5) Select best individuals $P^{sel}$ from $P$ based on their fitness values
        (6) Calculate the mean $\mu$ and covariance $\Sigma$ of $P^{sel}$
        (7) Use the $\mu$ and $\Sigma$ to sample new population, $P^{new}$
        (8) $P \leftarrow P^{new}$
**Until Termination criteria are met**
**Output:** $P$
```

We have not yet specified how to determine the d-dimensional box constraints that correspond to the given D-dimensional ones. Given some box constraints in $\mathbb{R}^D$, the following theorem gives the required box constraints for the search in $\mathbb{R}^d$.

Theorem 6.3.1. Let $f : \mathbb{R}^D \to \mathbb{R}$ be a function with intrinsic dimension $d_i < d < D$ that we want to optimise subject to the box constraint $\mathcal{X} \subset \mathbb{R}^D$, where $\mathcal{X}$ is centered around

0. Denote the intrinsic subspace Υ, and let Φ be a $D \times d_i$ matrix whose columns form an orthonormal basis for Υ. Denote by $x_t^* \in \Upsilon \cap \mathcal{X}$ an optimiser of f inside Υ. Let R be a $D \times d$ random matrix with independent standard Gaussian entries. Then there exists an optimiser $y^* \in \mathbb{R}^d$ such that $f(Ry^*) = f(x_t^*)$ w.p. 1, and for any choice of $\varepsilon \in (0,1)$, if $d > (\sqrt{d_i} + \sqrt{2\ln(1/\varepsilon)})^2$, then $||y^*||_2 \leqslant \frac{||x_t^*||_2}{\sqrt{d}-\sqrt{d_i}-\sqrt{2\ln(\frac{1}{\varepsilon})}}$ with probability at least $1-\varepsilon$. $\square$

Proof: The existence of y^* is guaranteed by Theorem 6.2.1 and the illustration of the fact is shown in figure 6.1. The global optimisers outside the subspace Υ are irrelevant since the function takes all its range of values in Υ. So our focus is to upper bound the length of y^*.

From the proof of Theorem 6.2.1 in [82] and the illustration in figure 6.1, we know that $\exists y^* \in \mathbb{R}^d$ s.t.

$$\Phi\Phi^T R y^* = x_t^* \tag{6.1}$$

Hence,

$$||x_t^*|| = ||\Phi\Phi^T R y^*|| \geq s_{\min}(\Phi\Phi^T R)||y^*|| \tag{6.2}$$

where we use the Rayleigh quotient inequality, and $s_{\min}(\cdot)$ denotes the smallest singular value.

Note that $\Phi\Phi^T R$ is a $d_i \times d$ random matrix with i.i.d. Gaussian entries. When $d = d_i$ it is a square matrix, and a bound on its smallest singular value was applied in [82]. Instead, for the case $d > d_i$ we employ the bound of Davidson & Szarek that applies to rectangular Gaussian matrices [20]. We have for any $\varepsilon \in (0,1)$ for which $\sqrt{d} - \sqrt{d_i} - \varepsilon > 0$, that:

$$||y^*|| \leqslant \frac{||x_t^*||}{\sqrt{d} - \sqrt{d_i} - \varepsilon} \tag{6.3}$$

with probability $1 - \exp(-\frac{\varepsilon^2}{2})$. Now setting $\exp(-\varepsilon^2/2) = \tau$ and solving for ε we get $\varepsilon = \sqrt{2\ln(\frac{1}{\tau})}$. Plugging this back and renaming τ to ε completes the proof. ■ $\square$

In Figure 6.4 we plotted the bound on the search box from our Theorem 6.3.1 for

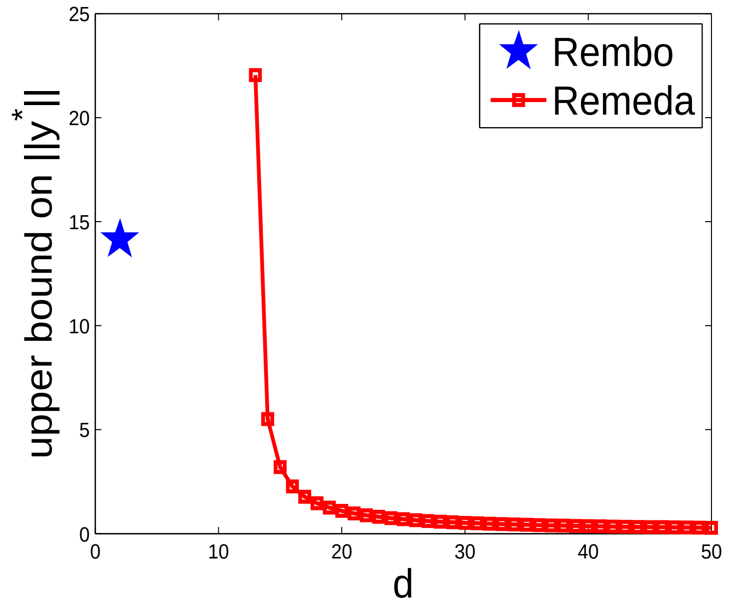


Figure 6.4: Comparison of our theoretical bound (Remeda), with various values of $d > d_i$ versus the bound of [82] (Rembo), which holds when $d = d_i$.

.

various values of $d > d_i$ in comparison with the bound in [82] for $d = d_i$. We see that our result is tighter for nearly all values of d and it explains why a smaller search box is sufficient when $d > d_i$. The single point for Rembo in figure 6.4 is for $d = d_i$ where as the curve for Remeda is for values of $d > d_i$.

In practice, of course, we typically have no knowledge of the value of d_i, in which case we cannot use theoretical bounds directly to set our search box. However, we can fix the search box – for instance to $\sqrt{d}$-times the coordinates of the original box, as suggested in [82], and our Theorem 6.3.1 then suggests that increasing d can eventually make this fixed-size box sufficiently large to contain the optimiser y^*. This is what we used in the experiments reported.

6.4 Experiments

Usually, when too small a population is used, evolutionary algorithms (EA) are not able to find the optimum. When you use too large a population, EA finds the optimum, but wastes some resources. Thus, in EA, we usually search for the smallest population size, for which the EA works.

Here we would like to find the optimal population size as function of the lower internal dimension, d of the problem. The hypothesis is that the optimal population size increases as the d increases and vis versa. Therefore, to test our hypothesis, we go by the following settings:

6.4.1 Parameter settings

To obtain the minimally required population size, a bisection search was used to find the population size that results in the lowest number of evaluations required to reach a predetermined value to reach (VTR). We start from a very large population size that can solve the problem within a large predetermined budget and then search for a small population size for which you cannot solve the problem anymore. In between these limits, we use binary search to find the optimal population size. This optimal population size is expressed as a function of d.

We use similar settings in our scalability experiment to see how scalable our method is as d_i increases. We also conducted some experiments to determine the dimension d.

6.4.2 Test functions and performance measures

We created test functions with low intrinsic dimension 5 from existing benchmark functions, by embedding the d_i-dimensional versions of these problems into higher D-dimensions. That is, we add $D - 5$ additional dimensions which do not impact on the function value,

Table 6.1: Test functions of low intrinsic dimension of 2 or 5. o is the shift vector.

PN	Name	Expression
1	Sphere	$\sum_{j=1}^{d_i}(x_j - o_j)^2$
2	Ackley's	$20 - 20\exp(-0.2\sqrt{\frac{1}{d_i}\sum_{j=1}^{d_i}((x_j - o_j) * M)^2})$-
		$\exp(\frac{1}{d_i}\sum_{j=1}^{d_i}(\cos(2\pi(x_j - o_j) * M))$+e
3	Elliptic	$\sum_{j=1}^{d_i}(10^6)^{\frac{j-1}{d_i-1}} * (x_j - o_j) * M)$
4	Rosenbrock	$\sum_{j=1}^{d_i-1}(100(z_j^2 - z_{j+1})^2 + (z_j - 1)^2)$
		$z = x - o + 1$
5	Branin	$(-1.275\frac{x_1^2}{\pi^2} + 5\frac{x_1}{\pi} + x_2 - 6)^2$
		$+(10 - \frac{5}{4\pi})\cos(x_1) + 10$

and (optionally) rotate the search space around the origin in a random direction. Hence, the functions will take D-dimensional inputs, but only 5 linear combinations of these input variables determine the function value. The algorithm will have no knowledge of which these directions are, not even that there are 5, but it has knowledge that the number of important directions is much less that D. The functions we employed in this way here are the following: Shifted sphere, Shifted Schwefel's problem 1.2, Shifted Rotated High Conditional Elliptic function, Shifted Ackley's function and Shifted Rosenbrock function. We also took the Branin function from [82] which has low intrinsic dimension 2. The functions are listed in table 6.1. In table 6.1, Functions 1-4 have low intrinsic dimensions of 5, while function 5 has low intrinsic dimension of 2.

We employ two common performance indicators:

(i) The fitness gap achieved under a fixed budget is the difference between the best fitness achieved and the true optimum; (ii) The scalability is the budget of function evaluations needed to reach a pre-defined value to reach.

6.5 Results and Discussion

6.5.1 Experiments on a $d_i = 2$ problem

In the first set of experiments we consider the $D = 25$ dimensional Branin function that has low intrinsic dimension $d_i = 2$. Though, we should note that D can be as large in principle, as we like since the working strategy and the budget usage of REMEDA are independent of D. In this experiment, we vary the internal working dimension d, and the population size N, under a fixed budget of 500 function evaluations.

Table 6.2: Fitness gap achieved by REMEDA on the Branin function ($d_i = 2$ embedded in $D = 25$), with a total budget of 500 function evaluations.

Pop. size	d=2		d=4		d=6	
	Mean	std	Mean	std	Mean	std
300	1.4297	2.601	2.4908	3.1013	3.9007	3.0322
150	0.4128	0.6607	1.1701	1.313	2.1368	2.1046
80	0.826	2.9973	0.4193	0.4459	0.8303	0.9331
40	0.6375	2.9073	0.04	0.0969	0.1927	0.3865
30	0.6737	2.4939	0.0336	0.0853	0.1038	0.2615

The results are shown in table 6.2, as obtained from 50 independent repetitions of each experiment. We can see from table 6.2 that $d = d_i = 2$ is not the best choice, as the size of the search box is not sufficient at $d = d_i$. Also observe that increasing d beyond 4 drops the performance – this is because searching in a larger dimensional space is not necessary and is less resource-effective. Furthermore, we see for all d tested, the higher the population sizes, the worse the performance. This is because a large population unnecessarily uses up the budget when the search only happens in a small dimensional subspace. With these insights in place, next we carry out a more comprehensive study.

6.5.2 Results and comparisons on problems with $d_i = 5$

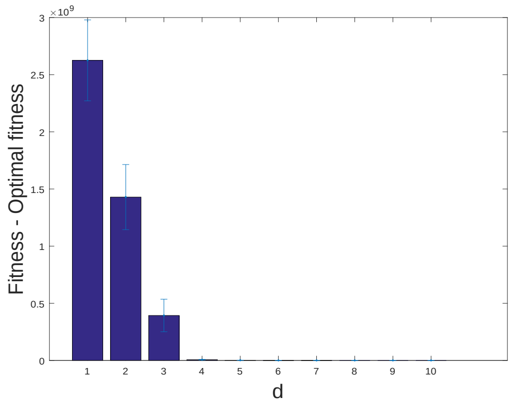
In this section, we compare our method with state of the art approaches in heuristic optimisation, on problems with low intrinsic dimension $d_i = 5$. The ambient dimension was $D = 1000$ in these experiments, but as already mentioned this can be much higher without causing problems as long as d_i stays low.

We expect that REMEDA should gain advantage from its ability to exploit low intrinsic dimensional property while other methods have not been designed to make use of such structure. On the other hand, REMEDA needs to find a good value for its internal dimension d without knowing d_i (as this information is normally not available in practice). This will use up part of the budget, but the hope is that it will pay off by a speedy progress in the search.

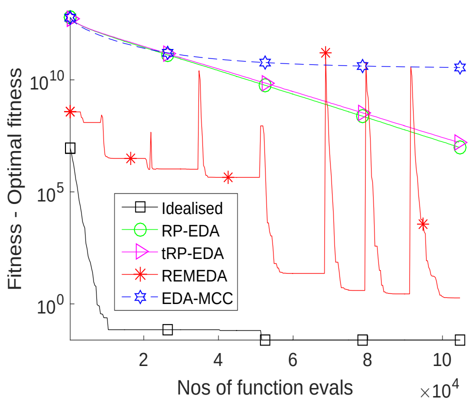
We start with $d = 1$, using convergence as a stopping criterion, and move up progressively to higher values of d until the fitness reached upon convergence is no longer improved by the increase of d. Within each value of d tried, we run REMEDA to convergence, until the relative change in fitness is below a threshold: $\frac{fv(g)-fv(g+1)}{fv(g)} < 10^{-8}$, where g is the generation count and fv is the fitness value. When this condition is satisfied, we move on to the next value of d, and re-initialise the population randomly (although other schemes could also be investigated). This is the reason we see the spikes in REMEDA in figure 6.6. The total number of fitness evaluations used throughout this process is the total budget that we then provide to the competing algorithms.

The bar chart in the leftmost plot of Figure 6.5 shows an example of the fitness gaps achieved with consecutive values of d. The error bars show one standard error from 25 independent repetitions. In the rightmost plot we show the evolution of the best fitness, averaged over 25 independent repetitions, on the run with the selected d (as described above). Superimposed, we also show the trajectories of competing state of the art methods: EDA-MCC [22], RP-EDA [47], and tRP-EDA [74]. All use the same budget– The unequal lengths of these trajectories are due to the various methods using part of their budget for setting their internal parameters. Now, to avoid the unequal lengths of the trajectories as shown in figure 6.5, we also plot the fitness gap against the total fitness evaluations used. See figure 6.6

From Figures 6.5 and 6.6, we can see that REMEDA attains a fitness value close to the optimum efficiently, while the other methods are not able to achieve the same within the same budget. We also superimposed an idealised version of EDA (EDA on low dimensional version of the problem, where low dimension = the unknown intrinsic dimension) – that is EDA that receives the d_i-dimensional version of the problem – and we see that REMEDA has almost the same performance as the idealised version of EDA.



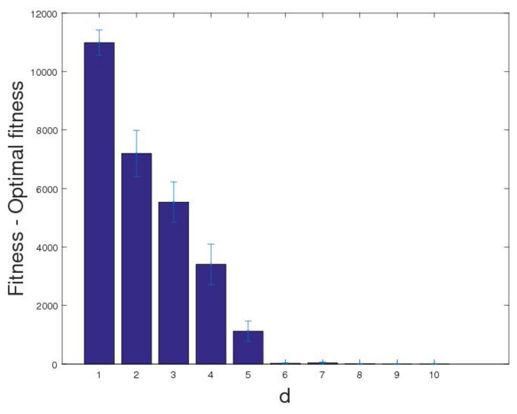
(a) Shifted Rosenbrock Function, Chosen $d = 9$



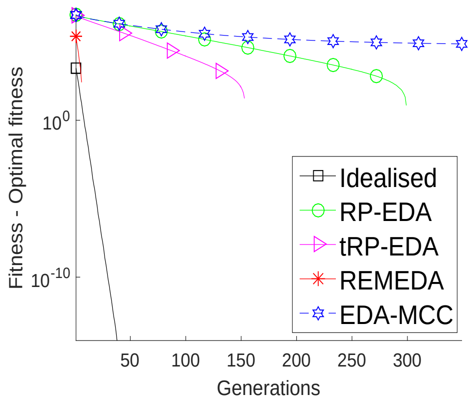
(b) Shifted Rosenbrock Function

Figure 6.6: Finding d (left) and evolution of best fitness (right) for REMEDA, 3 competing methods, and a d_i-dimensional EDA on the idealised problem. The figure on the right is plotted with Gap against no. of fitness evaluations. Results are averaged over 25 independent runs.

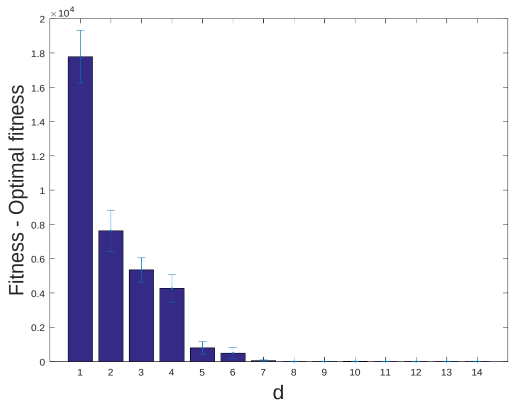
A comprehensive comparison is presented in Table 6.3, which also includes comparison with sep-CMA-ES [71] as one of the best state of the art. Bold font indicates statistically significant out-performance. Friedman Test was used in computing the statistical test.



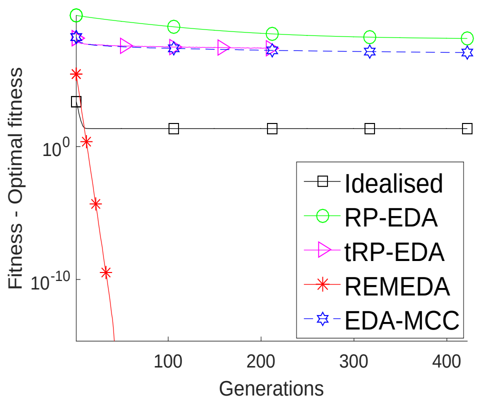
(a) Shifted Sphere Function, Chosen $d = 7$



(b) Shifted Sphere Function



(c) Shifted Schwefel's Problem 1.2, Chosen $d = 9$



(d) Shifted Schwefel's Problem 1.2

Figure 6.5: Finding d (left) and evolution of best fitness (right) for REMEDA, 3 competing methods, and a d_i-dimensional EDA on the idealised problem. Results are averaged over 25 independent runs.

From table 6.3, REMEDA has out-performed all of the state of the art methods on all the problems significantly. All the competing algorithms where given the same test suite. They were all given problems with d_i dimensions embedded in a 1000 dimensional, but REMEDA has the capability of exploiting the d_i dimensional structures of these test suites, while the competing methods are not able to. This shows that exploring and exploiting the structure of functions with low intrinsic dimension and searching in a lower dimensional subspace instead of the ambient dimensional space is very crucial in optimisation, thus confirming our hypothesis. In future, we would like to estimate the exact intrinsic dimensions of such type of problems and existing methods such is [5] and the references there could serve as a starting point.

Table 6.3: Comparing REMEDA with other state of the art methods.

Fn	REMEDA		sep-CMA-ES		EDA-MCC		tRP-ENS-EDA		RP-ENS-EDA	
	Mean	std	Mean	std	Mean	std	Mean	std	Mean	std
F1	**0**	**0**	3.81E+04	2.16E+04	6.41E+04	8.21E+03	37.80	3.17	25.79	2.05
F2	**0**	**0**	1.00E+07	7.92E+05	1.99E+06	1.42E+06	1.40E+08	7.99E+06	1.38E+08	6.59E+06
F3	**0.18**	**0.91**	4.75E+07	3.47E+06	2.90E+09	2.29E+08	6.81E+08	5.60E+07	2.84E+08	2.62E+07
F6	**45.92**	**216.11**	5.44E+06	2.09E+06	3.56E+10	4.93E+09	1.60E+07	1.88E+06	9.64E+06	1.44E+06
F8	**14.19**	**7.89**	21.67	0.01	21.34	0.06	21.66	0.01	21.43	0.06

6.5.3 Scalability experiments

Function evaluation is costly in most practical problems. Here we study what is the required budget of function evaluations to reach a specified value of the fitness gap.

Before running these scalability experiments, we carried out some experiments to determine the required population size as a function of the low intrinsic dimension of the problem, so that we can vary the latter and set the population size automatically. For this we use a bisection method as in [10]. To find the population size that results in the lowest number of evaluations required to reach a predetermined (VTR), we start from a very large population size that can solve the problem within a large predetermined budget and then search for a small population size that cannot solve the problem anymore. In between these limits we use binary search to find the optimal population size. We repeated this 25 times and took the average.

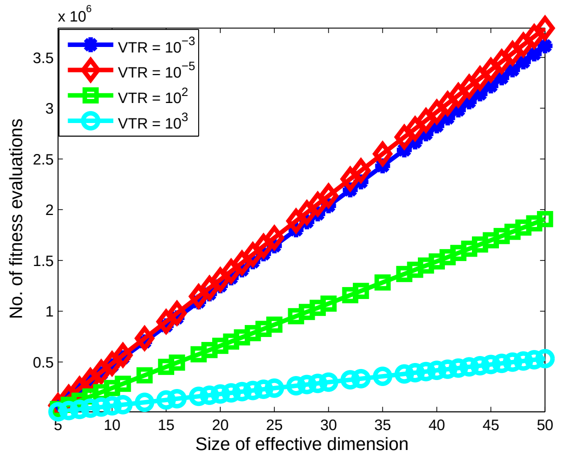
We fix the value to reach (VTR) to 10^{-5}, and vary the low intrinsic dimensionality of the problem $d_i \in [2, 50]$. We count the number of fitness evaluations needed for our proposed REMEDA to reach the VTR. The same experiment was repeated for three other choices of VTR: 10^{-3}, 10^2 and10^3 in order to make sure that the conclusions will not be specific to a particular choice of the VTR. In all these experiments the maximum fitness evaluations was fixed to 6×10^6, so the algorithm stops when the budget is exhausted or upon convergence.

Figure 6.7 shows the average number of function evaluations as computed from the successful runs out of 25 independent repetitions for each problem and each low intrinsic dimension tested. The success rates are as follows: For $VTR = 1e-3$, success rate = 86.359%, while that for $VTR = 1e-5$, $1e2$ and $1e3$ are 86.2564%, 93.9487% and 97.4359% respectively for the shifted sphere function. For the shifted Rastrigin function, $VTR = 1e-3$, success rate = 95.2%, $VTR = 1e-5$, success rate = 95.2% $VTR = 1e2$, success rate = 100% $VTR = 1e3$, success rate = 100%. For Shifted Ackley function, $VTR = 1e-3$, success rate = 72.9091% $VTR = 1e-5$, success rate = 69.0909% $VTR = 1e2$, success rate = 92.3636% $VTR = 1e3$, success rate= 92.5455%. For the Shifted Elliptic function, $VTR = 1e-3$, success rate = 11.4146% $VTR = 1e-5$, success rate = 11.0244% $VTR = 1e2$, success rate = 97.7561% $VTR = 1e3$, success rate = 97.7561%. From the figure, we observe a linear fit on scalability measurements.

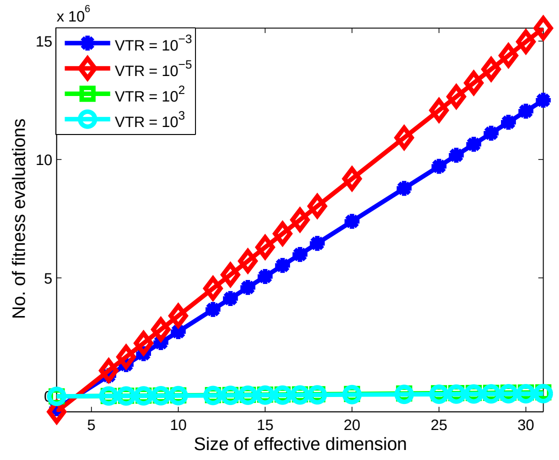
6.6 REMEDA with sub-Gaussian Random Embeddings

Sub-Gaussian distributions are those distributions whose tail decays no slower than that of the Gaussian [14]. It is a large class of distributions, and includes uniform bit-flipping and sparse distributions that allow fast matrix-vector multiplication. The theory presented in our earlier sections was developed for Gaussian embedding. Here we investigate the possibility to use random matrices R with i.i.d. sub-Gaussian entries instead – specifically the mentioned two random matrices, originally proposed by [3] for efficient random projections.

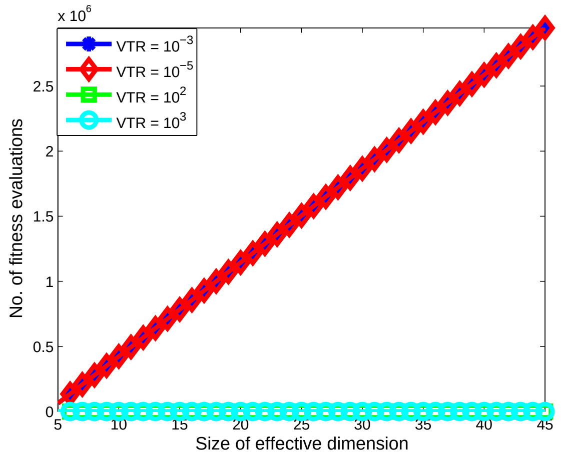
The first question is whether the optimum of a D-dimensional function with low



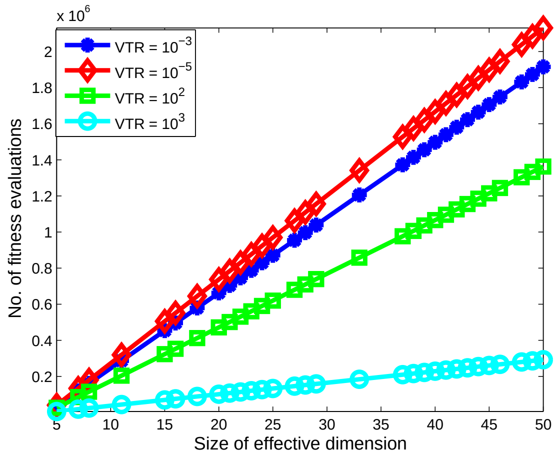
(a) Shifted Sphere Function



(b) Shifted Ackley Function



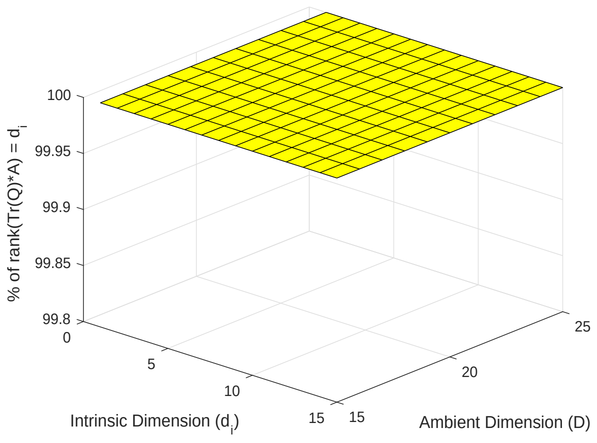
(c) Shifted Elliptic Function



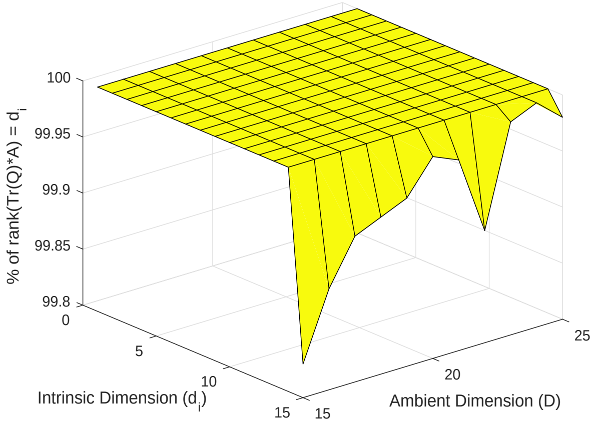
(d) Shifted Rastrigin Function

Figure 6.7: Number of function evaluations taken by successful runs of REMEDA to reach a pre-specified value to reach (VTR) as the problem low intrinsic dimensionality is varied in $d_i \in [2, 50]$. The markers represent averages computed from 25 independent repetitions.

intrinsic dimension d_i can still be found by searching in a $d << D$ dimensional random subspace? With Gaussian embedding, Theorem 1 of [82] guaranteed this. However, a crucial point in its proof was that the rank$(\Phi^T R)$ is no smaller than d_i with probability 1. This is also required for eq.(6.1). The measure theoretic arguments that ensured this in the case of Gaussian embedding no longer apply when we draw the entries of R from a discrete distribution. Indeed, now the rank$(\Phi^T R) = d_i$ only holds with high probability. Analytically computing this probability would be awkward, therefore, we experimentally estimate the rank of $\Phi^T R$ by Monte Carlo trials, and these results are shown in figure 6.8. We see that the rank very rarely falls below d_i, only for the sparse R, and only when the parameter d_i approaches d. From figure 6.8, as the intrinsic dimension approaches the ambient dimension, the rank $= d$ setting fails. The intuition is the probability of succeeding becomes small.



(a) Bit-flipping sub-Gaussian



(b) Sparse sub-Gaussian

Figure 6.8: Percentage of trials that satisfies rank$(\Phi^T R) = d_i$ out of 10,000 independent trials.

We then proceed to extend our Theorem 6.3.1 to the sub-Gaussian case in the following corollary.

Corollary 6.6.0.1 (to Theorem 6.3.1). Under the same conditions and with the same notations as in Theorem 1, and if R is a $D \times d$ random matrix with independent sub-Gaussian entries, then we have $||y^*|| \leqslant \frac{||x_t^*||}{\sqrt{d} - c\sqrt{d_i} - \sqrt{\frac{1}{c}\ln(\frac{1}{\varepsilon})}}$ with probability at least $1 - \varepsilon$, where $c > 0$ is a constant. $\square$

The proof is identical to that of Theorem 1, except in the step from eq.(6.2) to eq. (6.3), we use analogous result that bounds the smallest singular value of a sub-Gaussian matrix from [79] Theorem 5.39.

Proof: From the proof of Theorem 6.2.1 in [82], we know that $\exists y^* \in \mathbb{R}^d$ s.t.

$$\Phi\Phi^T R y^* = x_t^* \tag{6.4}$$

By Rayleigh quotient inequality, we have

$$||x_t^*|| = ||\Phi\Phi^T R y^*|| \geq s_{\min}(\Phi\Phi^T R)||y^*|| \tag{6.5}$$

where $s_{\min}(\cdot)$ denotes the smallest singular value.

Now bounding a rectangular matrix, $d > d_i$ as in our case, we employ theorem 5.39 from vershynin [79].

For every $\varepsilon \geqslant 0$, for which $\sqrt{d} - c\sqrt{d_i} - \varepsilon > 0$, we have

$$||y^*|| \leqslant \frac{||x_t^*||}{\sqrt{d} - c\sqrt{d_i} - \varepsilon} \tag{6.6}$$

with probability of atleast $1 - \exp(-c\varepsilon^2)$. Now setting $\exp(-c\varepsilon^2) = \tau$ and solving for ε we get, $\varepsilon = \sqrt{\frac{1}{c}\ln(\frac{1}{\tau})}$. Plugging this back and renaming τ to ε gives us,

$$||y^*|| \leqslant \frac{||x_t^*||}{\sqrt{d} - c\sqrt{d_i} - \sqrt{\frac{1}{c}\ln(\frac{1}{\varepsilon})}} \tag{6.7}$$

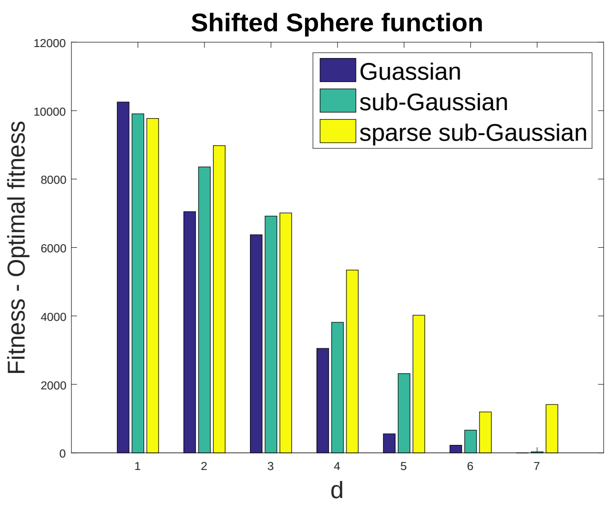
Hence the proof. □

Since the constant c has appeared in Corollary 1 (which was 1 in the Gaussian case), we may expect that the d-dimensional box guaranteed to contain the optimum may be larger than it was in the Gaussian case – therefore if we search in the same box sizes we might find that a larger d will be necessary.

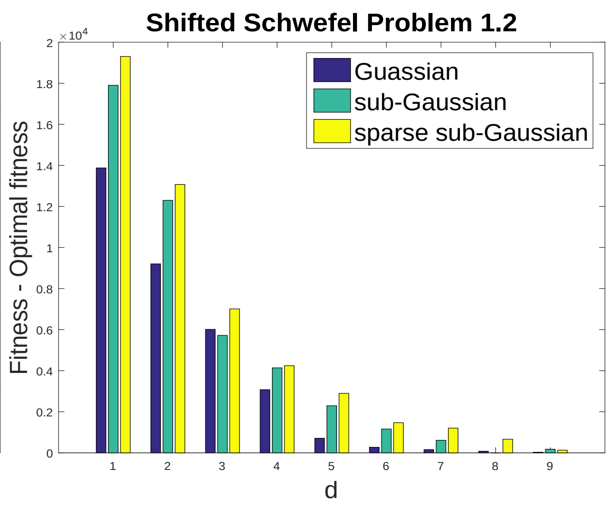
To test this, and to assess the practical potential of using sub-Gaussian embeddings, we repeated our previous experiments of searching for the value of d using the above mentioned two sub-Gaussian random matrices, and plotted the results in comparison with the Gaussian embedding in Figure 6.9. We can see that indeed the Sub-Gaussians were slightly worse than the Gaussian and needed a slightly larger d, which means they may require slightly more budget. But the differences are not large in practice, and a more clever searching strategy for setting d may easily resolve this. For instance, on the shifted sphere function the best d chosen by the Gaussian Random matrix is $d = 7$; the bit-flipping sub-Gaussian selected $d = 9$, and the sparse sub-Gaussian selected $d = 12$. For Shifted Schwefel's Problem 1.2 the selected ds by the Gaussian was $d = 9$, the sub-Gaussians chose $d = 10$, and $d = 11$ respectively. Clearly, the sub-Gaussian embeddings remain competitive, and their advantage is fast matrix-vector multiplication, which can be an asset when the ambient dimension D is very large, or when a speedy approximate solution to the problem is sought.

6.7 Summary

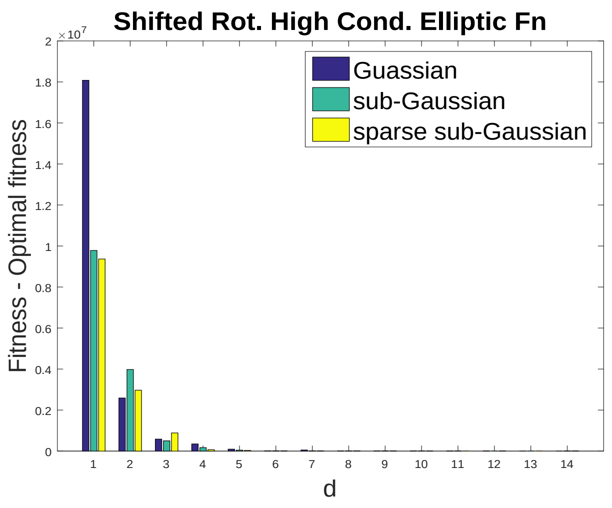
We proposed random embedding in Estimation of Distribution Algorithm to scale up EDA by exploiting the low intrinsic dimension of problems whereby the search takes place in a much lower dimensional space than that of the original problem. Our method is suited for large scale problems that take a large number of inputs but only depend on a few linear combinations of them. On such problems we have demonstrated that our method outperforms the state of the art algorithms in evolutionary computation. Our technique and its theoretical basis are applicable in principle to any optimisation method, and in the light that problems with low intrinsic dimension are quite prevalent in real-world applications, it seems a worthwhile avenue for future work to make use of it more widely.



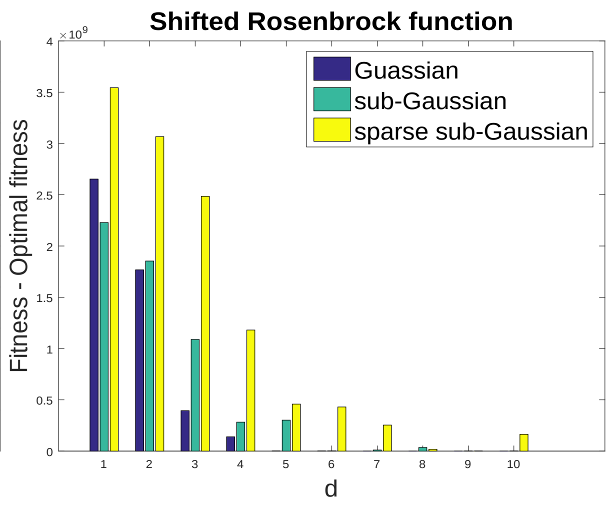
(a) Shifted Sphere Function



(b) Shifted Schwefel Problem 1.2



(c) Shifted Rot. high Cond. Elliptic



(d) Shifted Rosenbrock Function

Figure 6.9: Selecting d in the case of sub-Gaussian vs. Gaussian embeddings. Results are averaged over 25 independent runs.

CHAPTER 7

Conclusion and Future Work

The thesis presents approaches that are capable of searching in large dimensional continuous domains and investigates the capabilities of heavy tails search distributions with a clarification on a controversy that existed in the literature about the capabilities of Gaussian versus Cauchy search distributions.

In our framework called MCEDA in chapter 3, we address the susceptibility of Gaussian search distribution to premature convergence by using a heavy tail distribution called Cauchy as search distribution instead of Gaussian since it has the ability of escaping the early convergence due to its heavy tail and blend it together with the advantages of multivariate modelling to efficiently explore the search space. From extensive experiments conducted on 16 benchmark functions comparing MCEDA with MGEDA, we found that the long jumps of MCEDA due to its heavy tail paid dividend by outperforming MGEDA when the population is far from the global optimum. We also found that our method is able to work even with small population sizes. In comparing multivariate Cauchy EDA with univariate Cauchy EDA, we found that the former was superior to the later, thus bring in light the necessity of using Cauchy in a multivariate setting.

In chapter 4, we extended the comparison of Cauchy vs. Gaussian to high dimensional search. The rationale was to check if our finding in the previous comparison will hold in high dimensions and to resolve the controversy surrounding the merits of Cauchy search distribution. We conducted a large empirical study to benchmark the performance of Cauchy and Gaussian search distributions in EDA using a scalable black-box EDA optimizer. Our empirical results suggest that Cauchy search distributions perform badly, especially in high-dimensional spaces. We then developed theory that explains why large search steps are inefficient in high dimensional search spaces, and we showed that this inefficiency is indeed unavoidable in practice. We argued that a Gaussian search distribution has an in-built prioritizing strategy that implicitly focuses resources within a generation on selecting good search directions which Cauchy lacks. Based on our theoretical insights and understanding of high dimensional domains, a new framework was proposed from a minor modification to the standard Gaussian EDA which enforces search within a generation to take place at a fixed radius of the current population centre. This results in a remarkable improvement in high dimensional search.

Based on the good performances registered as a results of using heavy tail distribution in chapters 3 and 4(in the low dimensional regime), we decided to devise a new approach for high dimensional continuous black-box optimization by building on a recently proposed Random Projection Ensemble based EDA. Our focus was on the utility of the heavy tailed distributions when employed to sample the elements of our Random Projection matrices and named our approach t-Random Projection ensemble based EDA (tRP-ENS-EDA). Our results from extensive experiments we carried out to test our new approach suggest that taking Random Projection matrices with entries drawn with i.i.d t-distribution increases exploration and at the same time maintains exploitation and focus. We have shown that our method in comparison with a number of state of the art methods is very competitive and superior at times on 1000 dimensional problems, which warrants this method to be given the much needed attention it deserves by the experts/researchers in this community.

Finally, in our effort to remedy the curse of dimensionality Evolutionary Algorithms, especially the EDA type methods face, we proposed the random embedding technique in Estimation of Distribution Algorithm to scale up EDA for problems that have a low intrinsic dimension by taking into consideration only the important directions of problems. In this way, our search for the global optimum takes place in a much lower dimensional space than that of the original problem. This method takes a large number of inputs but only considers a few linear combinations of them. It can optimise problems of up to very high dimensions as long as their intrinsic dimensions are low. On such problems we have demonstrated that our method outperforms the state of the art algorithms which do not utilise the intrinsic structure of these problems in evolutionary computation. Our technique and its theoretical basis are applicable in principle to any optimisation problem with much higher ambient dimensions, provided sufficient computing resources are available.

In Future, we would like to fully evaluate the promise of our new EDA with uninform search distribution on a hypersphere method in chapter 4 and further develop it. Another important direction for future work will be to extend and generalise our methodology to other evolutionary search methods in order to investigate the generality of our conclusion beyond EDAs. We will also focus on further developing the method highlighted in chapter 5, by adapting other parameters such as the dimension of the reduced space, k. Due to the fact that problems with intrinsic dimensions are quite prevalent in real-world applications, we think it is a worthwhile avenue to venture into for future work to make use of it more widely. Therefore, we would like to focus on utilising the intrinsic dimensions of such type of problems in our future methods. We shall be working on developing a more clever approach of search for the best d that guarantee the existence of the global optimum in the d dimensional box for problems with higher intrinsic dimensions than the 5 and 2 considered here. We shall also endeavour to estimate the intrinsic dimensions of such type of problems just like it has been done in [15], [16] and [31]. The following paper [5] and the references there in, could serve as a starting point in estimating the intrinsic dimensions of such problem suite and project on to the intrinsic dimensional subspace and do the optimisation there instead of doing it in the Ambient space.

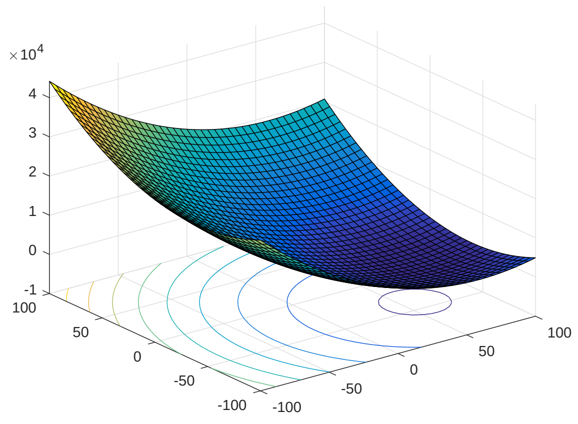
APPENDIX A

1

A.1 Some Box Constrained Test Problems from the CEC 2005 collection

A.1.1 Unimodal Functions

F_1: Shifted Sphere Function



(a) F_1: Shifted Sphere Function

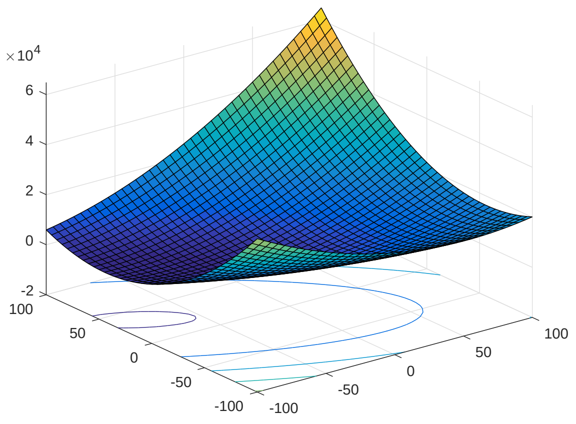
Definition: $F_1(X) = \sum_{j=1}^{D}(x_j - O_j)^2 - 450$,

where $O = \{o_1, ..., o_D\}$: Shifted global optimum, D: Dimension

Search Space: $-100 \leqslant x_j \leqslant 100, \;\; j = 1, ..., D$.

Global Optimum: $x^* = O$, $F_1(x^*) = -450$

F_2: Shifted Schwefel's Function 1.2



(a) F_2: Shifted Schwefel's Function 1.2

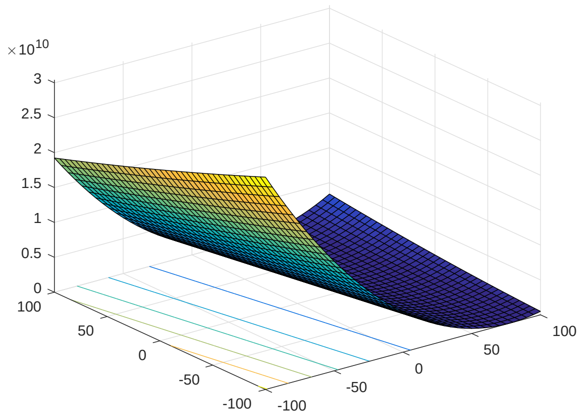
Definition: $F_2(X) = \sum_{i=1}^{D}(\sum_{j=1}^{i}(x_j - O_j)^2) - 450$,

where $O = \{o_1, ..., o_D\}$: Shifted global optimum, d: Dimension

Search Space: $-100 \leqslant x_j \leqslant 100, \;\; j = 1, ..., D$.

Global Optimum: $x^* = O$, $F_2(x^*) = -450$

F_3: Shifted Rotated High Conditioned Elliptic Function



(a) F_3: Shifted Rotated High Conditioned Elliptic Function

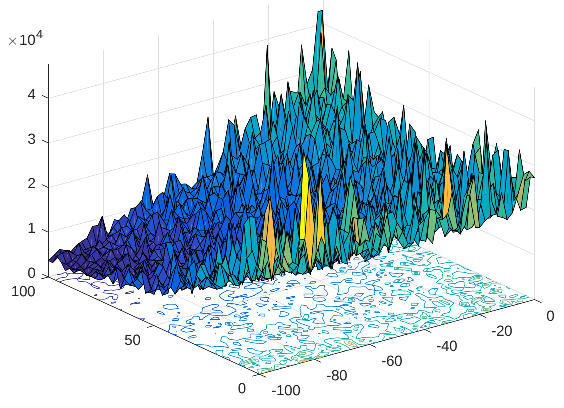
Definition: $F_3(X) = \sum_{j=1}^{D}(10^6)^{\frac{j-1}{D-1}}((x_j - O_j) \cdot M)^2 - 450$,

where $O = \{o_1, ..., o_D\}$: Shifted global optimum, D: Dimension, M: orthogonal matrix

Search Space: $-100 \leqslant x_j \leqslant 100, \;\; j = 1, ..., D.$

Global Optimum: $x^* = O$, $F_3(x^*) = -450$

F_4: Shifted Schwefel's Problem 1.2 with Noise in Fitness



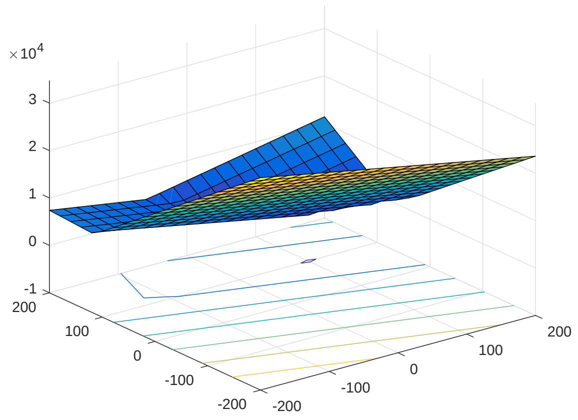
(b) F_4: Shifted Schwefel's Problem 1.2 with Noise in Fitness

Definition: $F_4(X) = (\sum_{i=1}^{D}(\sum_{j=1}^{i}(x_j - O_j)^2)) \cdot (1 + 0.4|N(0,1)|) - 450$, where $O = \{o_1, ..., o_D\}$: Shifted global optimum, D: Dimension, $X = \{x_1, ..., x_D\}$

Search Space: $-100 \leqslant x_j \leqslant 100, \ \ j = 1, ..., D.$

Global Optimum: $x^* = O$, $F_4(x^*) = -450$

F_5: **Schwefels Problem 2.6 with Global Optimum on Bounds**



(c) F_5: Shifted Schwefel's Problem 1.2 with Noise in Fitness

Definition: $F_5(X) = max\{|x_1+2x_2-7|, |2x_1+x_2-5|\}, i = 1, ..., N, x^* = [1, 3], F(x^*) = 0,$

$$F_5(X) = max\{|A_jX - B_j|\} - 310, \;\; i = 1, ..., D, \;\; x = [x_1, ..., x_D]$$

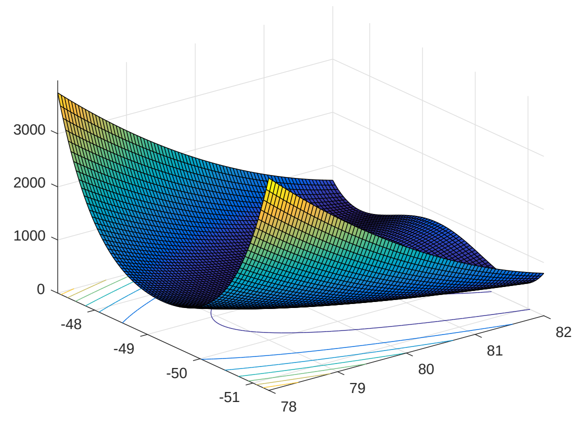
where $O = \{o_1, ..., o_D\}$: Shifted global optimum, D: Dimension, $A = D * D$ Matrix

Search Space: $-100 \leqslant x_j \leqslant 100, \;\; j = 1, ..., D.$

Global Optimum: $x^* = O$, $F_5(x^*) = -310$

A.1.2 Multimodal Functions

F_6: Shifted Rosenbrock Function



(d) F_6: Shifted Rosenbrock Function

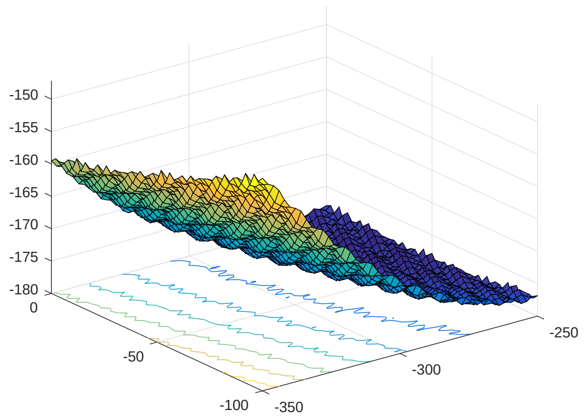
Definition: $F_6(X) = \sum_{j=1}^{D-1}(100((x_j - O_j + 1)^2 - (x_{j+1} - O_{j+1} + 1))^2 + ((x_j - O_j + 1) - 1)^2) + 390,$

where $O = \{o_1, ..., o_D\}$: Shifted global optimum, D: Dimension

Search Space: $-100 \leqslant x_j \leqslant 100, \ \ j = 1, ..., D.$

Global Optimum: $x^* = O$, $F_6(x^*) = 390$

F_7: Shifted Rotated Griewank Function without Bounds



(e) F_7: Shifted Rotated Griewank Function without Bounds

Definition: $F_7(X) = \sum_{i=1}^{D}(\frac{((x_j - O_j)*M)^2}{4000}) - \prod_{i=1}^{D} \cos(\frac{(x_j - O_j)*M}{\sqrt{i}}) + 1 - 180$,

where $O = \{o_1, ..., o_D\}$: Shifted global optimum, is outside of the initialization range

D: Dimension, $M^{'}$: linear transformation matrix, condition number $= 3$

$M = M^{'}(1 + 0.3|N(0,1)|)$

Search Space: $0 \leqslant x_j \leqslant 600, \;\; j = 1, ..., D.$

Global Optimum: $x^* = O$, $F_7(x^*) = -180$

APPENDIX B

1

B.1 Parameter Estimation of location(μ) and positive definite matrix(Σ) for Cauchy -Derivation

Multivariate Cauchy density function is defined as follows:

$$f(y;\mu,\Sigma,D) = \frac{\Gamma(\frac{1+D}{2})}{\Gamma(\frac{1}{2})\pi^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}[1+(y-\mu)^T\Sigma^{-1}(y-\mu)]^{\frac{1+D}{2}}} \tag{B.1}$$

Where y is the observed variable, D the dimension of the function, μ the location parameter and Σ a matrix value parameter that encodes pairwise dependencies.

If y is our observed variable, then the data log-likelihood is defined as follows:

$$L(\theta) = \sum_{n=1}^{N} log \int_u N(y_i|\mu, \frac{\Sigma}{u}).Ga(u|\frac{\nu}{2}, \frac{\nu}{2})du \tag{B.2}$$

Where u our unobserved or latent variable and θ is the parameter we want to estimate.

Our goal here is to fine the maximum likelihood estimate (MLE) of the parameter θ which represents μ and Σ.

We Let the posterior probability of u given y_i be:

$$P(u|y_i) = \frac{P(y_i|u).P(u)}{P(y_i)} \tag{B.3}$$

This is proportional to the product of the normal distribution and Gamma distribution as shown below: A convolution of Gaussian and Gamma Distribution

This can be written as

$$P(u|y_i) \propto N(y_i|\mu, \frac{\Sigma}{u}).Ga(u|\frac{\nu}{2}, \frac{\nu}{2}) \tag{B.4}$$

Now plugging in the probability density functions, we have

$= \frac{1}{(2\pi)^{D/2}|\frac{\Sigma}{u}|^{D/2}} exp(-\frac{1}{2}(y_i - \mu)^T(\frac{\Sigma}{u})^{-1}(y_i - \mu))\frac{(\frac{\nu}{2})^{\nu/2}(u)^{\nu/2-1}}{\Gamma(\frac{\nu}{2})} exp(-\frac{\nu}{2}u)$, D is dimension of the data and ν the degree of freedom.

$= \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}(u)^{-D/2}} exp(-\frac{1}{2}(y_i - \mu)^T(\Sigma)^{-1}u(y_i - \mu))\frac{(\frac{\nu}{2})^{\nu/2}(u)^{\nu/2-1}}{\Gamma(\frac{\nu}{2})} exp(-\frac{\nu}{2}u)$

$\propto (u)^{D/2}(u)^{\nu/2-1} exp(-\frac{1}{2}(y_i - \mu)^T(\Sigma)^{-1}u(y_i - \mu) - \frac{\nu}{2}u)$

$$\propto (u)^{\frac{D+\nu}{2}-1} exp((-\frac{1}{2}(y_i - \mu)^T(\Sigma)^{-1}u(y_i - \mu) - \frac{\nu}{2})u) \tag{B.5}$$

By comparing equation (21) with $f(u, \alpha, \beta) = \frac{\beta^\alpha u^{\alpha-1}}{\Gamma(\alpha)} e^{-\beta u}$, we have

$Ga(\frac{\nu+D}{2}, \frac{1}{2}[(y_i - \mu)^T\Sigma^{-1}(y_i - \mu) + \nu])$

$E(u_i) = \frac{\nu+D}{\nu+\delta(y_i,\mu,\Sigma)}$, where $\delta = \frac{1}{2}(y_i-\mu)^T\Sigma^{-1}(y_i-\mu)$ is the Mahalanobis distance from y to the center μ with respect to Σ. Now,

Now from (18), creating Expectation gives

$= \sum_{n=1}^{N} log \int_u Q_i(u).\frac{N(y_i|\mu,\frac{\Sigma}{u}).Ga(u|\frac{\nu}{2},\frac{\nu}{2})}{Q_i(u)}du$

$= \sum_{n=1}^{N} log E_{u\sim Q_i}[\frac{N(y_i|\mu,\frac{\Sigma}{u}).Ga(u|\frac{\nu}{2},\frac{\nu}{2})}{Q_i(u)}]$

Now for any Q_i s.t $\int Q_i(u)du = 1$ and $Q_i(u) \geqslant 0$, we have

$$= \sum_{n=1}^{N} log E_{u\sim Q_i}[\frac{P(y_i|u;\theta).P(u)}{Q_i(u)}] \geqslant \sum_{i=1}^{N} E_{u\sim Q_i}[log\frac{P(y_i,u,\theta)}{Q_i(u)}] \tag{B.6}$$

Equation (22) was obatined from Jensen's inequalities

$= \sum_{n=1}^{N} \int_u Q_i(u) log[\frac{N(y_i|\mu,\frac{\Sigma}{u}).Ga(u|\frac{\nu}{2},\frac{\nu}{2})}{Q_i(u)}]du$

This creates a lower bound on the function for any Q

$$= \sum_{n=1}^{N} E_{u\sim Q_i}[log\frac{N(y_i|\mu,\frac{\Sigma}{u}).Ga(u|\frac{\nu}{2},\frac{\nu}{2})}{Q_i(u)}] \tag{B.7}$$

Let

$$\frac{N(y_i|\mu,\frac{\Sigma}{u}).Ga(u|\frac{\nu}{2},\frac{\nu}{2})}{Q_i(u)} = C \tag{B.8}$$

For the Jensen's inequality in (22) to hold for equality, (24) should be a constant

So we need to choose Q so that $\frac{P(y_i,u,\theta)}{Q_i(u)}$ = Constant $\forall u$

To ensure this, the ratio. $Q_i(u) \propto P(y_i, u, \theta)$ should remain constant. Also

$\int_u Q_i(u) = 1$ and $Q_i(u) \geqslant 0$. Now the choice of

$Q_i(u) = \frac{P(y_i,u,\theta)}{\int_u P(y_i,u,\theta)}$ must sum to 1

$= \frac{P(y_i,u,\theta)}{P(y_i,\theta)} = P(u|y_i, \theta)$, by the definition of conditional probability.

Now in the E-step, we set, $Q_i(u) = P(u|y_i, \theta)$. So the E-step is

$Q_i(u) \propto Ga(u|\frac{\nu+D}{2}, \frac{1}{2}[(y_i - \mu)^T \Sigma^{-1}(y_i - \mu) + \nu])$

Now from (23), we have , $A = \sum_{n=1}^{N} E_{u \sim Q_i}[log \frac{N(y_i|\mu, \frac{\Sigma}{u}).Ga(u|\frac{\nu}{2}, \frac{\nu}{2})}{Q_i(u)}]$

Plugging in the pdf's, we have

$A = \sum_{n=1}^{N} E_{u \sim Q_i}[log(2\pi)^{-D/2}|\Sigma|^{-1/2}(u)^{D/2} - \frac{1}{2}(y_i - \mu)^T(\frac{\Sigma}{u})^{-1}(y_i - \mu) + log(\frac{\nu}{2})^{\frac{\nu}{2}} -$
$log(\Gamma(\frac{\nu}{2}))$
$+ log(u)^{\frac{\nu}{2}-1} - \frac{\nu}{2}u] - logQ_i(u)$

$A = \sum_{n=1}^{N}[E_{u_i \sim Q_i}(log(2\pi)^{-D/2}|\Sigma|^{-1/2}(u)^{D/2}) - \frac{1}{2}(y_i - \mu)^T\Sigma^{-1}(y_i - \mu)E(u) + E[log\frac{(\frac{\nu}{2})^{\frac{\nu}{2}}}{\Gamma(\frac{\nu}{2})}]$
$+ (\frac{\nu}{2} - 1)E(log(u)) - \frac{\nu}{2}E(u)] - E[logQ_i(u)]$

$$\frac{\partial A}{\partial \mu} = \sum_{i=1}^{N}[\frac{\partial(-\frac{1}{2}(y_i-\mu)^T\Sigma^{-1}(y_i-\mu)E(u_i))}{\partial \mu}] \tag{B.9}$$

$\frac{\partial(-\frac{1}{2}(y_i-\mu)^T\Sigma^{-1}(y_i-\mu)E(u_i))}{\partial \mu} = \frac{\partial(-\frac{1}{2}(y_i^T-\mu^T)(\Sigma^{-1}y_i-\Sigma^{-1}\mu)}{\partial \mu}$

$= -\frac{1}{2}(y_i^T-\mu^T)(-\Sigma^{-1}) + (\Sigma^{-1}y_i - \Sigma^{-1}\mu)(\frac{1}{2})$
$= [-\frac{1}{2}(-y_i^T\Sigma^{-1} + \mu^T\Sigma^{-1}) + ({y_i}^T\Sigma^{-1} - \mu^T\Sigma^{-1})\frac{1}{2}]^T$

$= [\frac{1}{2}y_i^T\Sigma^{-1} - \frac{1}{2}\mu^T\Sigma^{-1} + \frac{1}{2}{y_i}^T\Sigma^{-1} - \frac{1}{2}\mu^T\Sigma^{-1}]^T = [y_i^T\Sigma^{-1} - \mu^T\Sigma^{-1}]^T$

$$= \Sigma^{-1}y_i - \Sigma^{-1}\mu \tag{B.10}$$

now plugging (26) into (25), we have

$\frac{\partial A}{\partial \mu} = \sum_{i=1}^{N}[(\Sigma^{-1}y_i - \Sigma^{-1}\mu)E(u_i)]$

Now since $\frac{\partial A}{\partial \mu} = 0$, we have

$= \sum_{i=1}^{N}[(\Sigma^{-1}y_i - \Sigma^{-1}\mu)E(u_i)] = \sum_{i=1}^{N}\Sigma^{-1}y_iE(u_i) - \sum_{i=1}^{N}\Sigma^{-1}\mu E(u_i) = 0$

$\sum_{i=1}^{N}\Sigma^{-1}\mu E(u_i) = \sum_{i=1}^{N}\Sigma^{-1}y_iE(u_i)$

$\mu\sum_{i=1}^{N}E(u_i) = \sum_{i=1}^{N}y_iE(u_i)$

$$\mu = \frac{\sum_{i=1}^{N}y_iE(u_i)}{\sum_{i=1}^{N}E(u_i)} \tag{B.11}$$

From (A) again, taking the derivative w.r.t Σ^{-1}, we have

$\frac{\partial A}{\partial \Sigma^{-1}} = \sum_{i=1}^{N} [E_{u \sim Q_i} [\frac{\partial((log(2\pi)^{-D/2}|\Sigma|^{-1/2}(u)^{D/2}))}{\partial \Sigma^{-1}} - \frac{1}{2}\frac{\partial((y_i-\mu)^T\Sigma^{-1}(y_i-\mu))u}{\partial \Sigma^{-1}}]$

From the definition of $\frac{\partial log(detA)}{\partial A} = (A^{-1})^T = (A^T)^{-1}$ and by the chain rule, we can deduce that

$\frac{\partial log(detA)}{\partial A^{-1}} = \frac{\partial log(detA)}{\partial A} * \frac{\partial A}{\partial A^{-1}} = A^{-1} * \frac{\partial (A^{-1})^{-1}}{\partial A^{-1}}$. If $B = A^{-1}$ Then

$\frac{\partial log(detA)}{\partial A^{-1}} * \frac{\partial A}{\partial A^{-1}} = A^{-1} * \frac{\partial B^{-1}}{\partial B}$ and $A^{-1} * -\frac{1}{(A^{-1})^2} = -A^{-1} * A^2 = -A$

Now, $\frac{\partial log|\Sigma|^{-\frac{1}{2}}}{\partial \Sigma^{-1}} = -\frac{1}{2}\frac{\partial log|\Sigma|}{\partial \Sigma^{-1}} = -\Sigma * -\frac{1}{2} = \frac{1}{2}\Sigma$

Therefore, $\frac{\partial A}{\partial \Sigma^{-1}} = \sum_{i=1}^{N}[\frac{1}{2}\Sigma - \frac{1}{2}\frac{\partial(y_i-\mu)^T\Sigma^{-1}(y_i-\mu)E(u_i)}{\partial \Sigma^{-1}}]$

$\sum_{i=1}^{N}[\frac{1}{2}\Sigma - \frac{1}{2}(y_i-\mu)(y_i-\mu)^T E(u_i)] = 0$

$\frac{1}{2}\sum_{i=1}^{N}\Sigma = \sum_{i=1}^{N}\frac{1}{2}(y_i-\mu)(y_i-\mu)^T E(u_i)]$

$N\Sigma = \sum_{i=1}^{N}(y_i-\mu)(y_i-\mu)^T E(u_i)$

$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(y_i-\mu)(y_i-\mu)^T E(u_i)$ Hence,

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(y_i-\mu)(y_i-\mu)^T E(u_i) \tag{B.12}$$

List of References

[1] *"Large Scale Global Optimization 2010." CEC 2010 Special Session. http://nical.ustc.edu.cn/cec10ss.php*, Accessed on Sept. 10, 2014. 2 citations in sections 2.6 and 5.4.1.

[2] M. Abramowitz and I. A. Stegun (Eds.). Handbook of mathematical functions with formulas, graphs and mathematical tables (applied mathematics series 55). Washington, DC: NBS., 1964. 4 citations in sections 5.2, 5.2, 5.2, and 5.2.

[3] Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of Comp. & Sys. Sci*, 66 (4):pp. 671–687, 2003. 3 citations in sections 2.4, 2.4, and 6.6.

[4] D. Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Computer and Systems Sciences*, 66(4):671–687, 2003. 2 citations in sections 2.4 and 2.4.

[5] L. Amsaleg, O. Chelly, and T. Furon. Estimating local intrinsic dimensionality. In *KDD*, 2015. 2 citations in sections 6.5.2 and 7.

[6] R. Armañanzas, I. Inza, R. Santana, Y. Saeys, J.L. Flores, J.A. Lozano, Y. Van de Peer, R. Blanco, V. Robles, C. Bielza, and P. Larrañaga. A review of estimation of distribution algorithms in bioinformatics. *BioData Mining*, 2008. One citation in section 2.3.1.

[7] K. Ball. *An Elementary Introduction to Modern Convex Geometry. In: Flavors of Geometry.* MSRI Publications, 1997. One citation in section 4.5.1.

[8] R. Bellman. The theory of dynamic programming. 1955. One citation in section 2.3.2.

[9] J. Bergstra and Y. Bengio. Random search for hyperparameter optimization. *JMLR*, 13:281–305, 2012. One citation in section 6.1.

[10] P. Bosman. On empirical memory design, faster selection of bayesian factorizations and parameter-free Gaussian EDAs. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO), pp. 389-396. ACM*, 2009. One citation in section 6.5.3.

[11] P. Bosman and D. Thierens. An algorithmic framework for density estimation based evolutionary algorithms. Technical report, UU-CS., 1999. One citation in section 2.3.1.

[12] P. A.N Bosman and J. Grahl. Matching inductive search bias and problem structure in continuous estimation of distribution algorithms. *European Journal of Operational Research, pp. 1246-1264*, 185.3, 2008. One citation in section 2.

[13] P.A.N Bosman and T. Dirk. Numerical optimization with real-valued estimation of distribution algorithms. In *Scalable Optimization via Probabilistic Modeling, Springer Berlin Heidelberg, pp. 91-120.*, 2006. 2 citations in sections 2.8.4 and 2.8.4.

[14] V.V. Buldygin and Yu.V. Kozachenko. Subgaussian random variables. *Ukrainian Math*, 32:483 – 489, 1980. One citation in section 6.6.

[15] F. Camastra. Data dimensionality estimation methods: A survey. *Elsevier*, 2011. One citation in section 7.

[16] F. Camastra and A. Staiano. Intrinsic dimension estimation, pp 26-41. *Information Sciences*, 328, 2016. One citation in section 7.

[17] W.J. Conover. Practical nonparametric statistics. 1999. One citation in section 5.4.2.

[18] P. Constantine. Active subspace methods in theory and practice: applications to kriging surfaces. *SIAM J. Sci. Comput.*, pages pp. 1500–1524, 36(4). 2 citations in sections 1.3 and 6.1.

[19] S. Dasgupta and A. Gupta. An elementary proof of the johnson-lindenstrauss lemma. *Random Struct. Alg. 22 , pp 60Ű65.*, 2002. 2 citations in sections 2.4 and 2.4.

[20] K.R. Davidson and S.J. Szarek. Local operator theory, random matrices and banach spaces, in handbook of the geometry of banach spaces, vol 1. pp. 317-366., 2001. One citation in section 6.3.

[21] J. DemŽar. Statistical comparisons of classifiers over multiple data sets. *Machine Learning Research*, 7:1–30, 2006. One citation in section 5.4.2.

[22] W. Dong, T Chen, P. Tiño, and X. Yao. Scaling up estimation of distribution algorithm for continuous optimisation. *IEEE Transaction on Evolutionary Computation, Vol 17, Issue 6, pp. 797-822*, 2013. 10 citations in sections 1.2, 2.3.1, 2.8, 2.8.3, 4.1, 4.2, 4.3.1, 4.3.1, 6.1, and 6.5.2.

[23] W. Dong and X. Yao. Covariance matrix repairing in gaussian based edas. In *Congress on Evolutionary Computation (CEC)*, 2007. 2 citations in sections 2.4 and 1.

[24] W. Dong and X. Yao. Unified eigen analysis on multivariate gaussian based estimation of distribution algorithms. *Information Sciences*, 2008. 6 citations in sections 2.3.2, 2.8, 2.8.2, 2.8.2, 2.8.2, and 2.9.

[25] R. J. Durrant and A. Kabán. Error bounds for Kernel Fisher Linear Discriminant in Gaussian Hilbert space. In *Proc. of the 15th international Conference on Artificial Intelligence and Statistics (AISTATS), pp 337-345*, 2012. 2 citations in sections 4.5.2 and 4.5.2.

[26] R.J Durrant and A. Kaban. Random projections for machine learning and data mining: Theory and applications. Technical report, University of Birmingham, 2012. 2 citations in sections 2.4 and 2.

[27] R.J Durrant and A. kaban. A tight bound on the performance of fisher's linear discriminant in randomly projected data spaces. *Pattern Recognition Letters*, 33:pp 911 – 919, 2012. One citation in section 5.2.

[28] C. Echegoyen, Q. Zhang, A. Mendiburu, R. Santana, and J.A. Lozano. On the limits of effectiveness in estimation of distribution algorithms. 2011. One citation in section 2.9.

[29] A.E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transaction on Evolutionary Computationtion. Vol 3, Issue 2, pp. 124-141*, 1999. One citation in section 5.3.

[30] N. Hansen et al. Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. *ACM*, 2010. One citation in section 2.8.4.

[31] M. Fan, N. Gu, H.Qiao, and B. Zhang. Intrinsic dimension estimation of data by principal component analysis. *CoRR*, 2010. One citation in section 7.

[32] W. Feller. *An introduction to probability theory and its applications.*, volume 2. John Wiley & Sons, 2008. One citation in section 2.8.1.

[33] D. B. Fogel. What is evolutionary computation? *IEEE Spectrum, pp. 26-32.*, 2000. One citation in section 2.2.1.

[34] J. Friedman. An overview of predictive learning and function approximation. *NATO ASI Series of Computer and Systems Sciences, pp. 1-61*, 136, 1994. One citation in section 2.3.2.

[35] N Hansen. The cma evolution strategy: a comparing review. *Springer*, 2006. 2 citations in sections 2.8.4 and 2.8.4.

[36] N. Hansen. *Towards a New Evolutionary Computation.* Springer Berlin Heidelberg, 2006. One citation in section 2.8.

[37] N. Hansen. Benchmarking a bi-population CMA-ES on the BBOB-2009 function testbed. In *Proc. of Genetic and Evolutionary Computation (GECCO), pp. 2389-2396*, 2009. One citation in section 4.1.

[38] N. Hansen, F. Gemperle, A. Auger, and P. Koumoutsakos. When do heavy-tail distributions help? In *Proc. of the 9th International Conference on Parallel Problem Solving from Nature (PPSN), LNCS 4193, pp. 62-71*, 2006. 2 citations in sections 4 and 4.4.3.

[39] N Hansen, M. Hausschild, and M. Pelikan. An introduction and survey of estimation of distribution algorithms. *Swarm and Evolutionary Computation, Issue 3, pp. 111-128*, 1, 2011. One citation in section 2.2.

[40] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation, pp. 159-195*, 9.2, 2001. One citation in section 2.8.4.

[41] F. Hutter. *Automated Configuration of Algorithms for Solving Hard Computational Problems.* PhD thesis, 2009. One citation in section 6.1.

[42] C. Isbell J. De Bonet and P. Viola. Mimic: Finding optima by estimating. 1997. One citation in section 2.3.1.

[43] J.Sobieszczanski-Sobieski and R.T Haftka. Multidisciplinary aerospace design optimisation: Survey of recent developments. *Struct. optim*, 48:1–23, 2013. One citation in section 1.2.

[44] A. Kabán. New bounds for compressive linear least squares regression. In *The 17-th International Conference on Artificial Intelligence and Statistics (AISTATS 2014)*, 2014. One citation in section 3.

[45] A. Kabán. Non-asymptotic analysis of Compressive Fisher Discriminants in terms of the effective dimension. In *Proc. of the 7th Asian Conference on Machine Learning (ACML), Journal of Machine Learning Research-Proceedings Track, pp. 17-32*, 2016. One citation in section 4.5.2.

[46] A. Kabán, J. Bootkrajang, and R.J. Durrant. Towards large scale continuous eda: a random matrix theory perspective. In Christian Blum and Enrique Alba, editors, *GECCO*, pages 383–390. ACM, 2013. 8 citations in sections 1.2, 2.4, 2.4, 2.9, 5.2, 5.2, 5.2, and 5.4.2.

[47] A. Kabán, J. Bootkrajang, and R.J. Durrant. Towards Large Scale Continuous EDA: A Random Matrix Theory Perspective. *Evolutionary Computation, MIT Press*, 2016 (In Print). 8 citations in sections 2.8, 2.8.5, 2.8.5, 3.1, 4.1, 5.2, 6.1, and 6.5.2.

[48] O. Kramer. *Self-adaptive heuristics for evolutionary computation.*, volume 147. Springer Berlin Heidelberg., 2008. 2 citations in sections 2.8.4 and 2.8.4.

[49] P. Larranaga and J.A Lozano. *Estimation of Distribution Algorithms: A new tool for Evolutionary Computation.* Kluwer Academic Publishers, 2001. 4 citations in sections 1.2, 1.3, 2.2.2, and 3.1.

[50] P. Larranga and J.A. Lozano. *Estimation of Distribution Algorithms: A new tool for evolutionary computation.* Kluwer Academic Publishers, 2002. One citation in section 2.9.

[51] T. Sonada M. Hasenjager, B. Sendhoff and T. Arima. Three dimensional evolutionary aerodynamics design optimisation with CMA-ES. In *Genetic and Evolutionary computation conference*, pages 2173–2180, 2005. One citation in section 1.2.

[52] B. Newhouse M. Mahoney and G. Mukherjee. Algorithms for modern massive data set analysis. In *Lecture Notes.* 2009. 2 citations in sections 2.4 and 2.4.

[53] Y. Jin M. Olhofer and B. Sendhoff. Adaptive encoding for aerodynamic shape optimisation using evolutionary strategies. In *IEEE Congress on Evolutionary Computation*, volume 2, pages 576–583, 2001. One citation in section 1.2.

[54] D. Goldberg und F. Lobo M. Pelikan. A survey of optimization by building and using. *Computational Optimization and Applications*, 2002. 3 citations in sections 2.2, 2.2.1, and 2.2.1.

[55] Mei, Omidvar, Li, and Yao. Competitive divide-and-conquer algorithm for unconstrained large scale black-box optimization. *ACM Transactions on Mathematical Software (TOMS)*, 42(2):13, 2016. One citation in section 2.6.4.

[56] H. Muehlbrandt. Large scale optimization with estimation of distribution algorithms. Master's thesis, School of Computer Science, University of Birmingham, 2013. 3 citations in sections 2.3.1, 2.8.1, and 2.8.1.

[57] H. Muhlenbein and T. Mahnig. Convergence theory and application of the factorized distribution algorithm. *Computing and Information Technology*, 7:19–32, 1999. One citation in section 2.9.

[58] M. N. Omidvar and X. Li. A comparative study of CMA-ES on large scale global optimisation. In *Proc. AI 2010: Advances in Artificial Intelligence, pp. 303-312*, 2011. One citation in section 4.1.

[59] M.N.i Omidvar, X. Li, and K. Tang. Designing benchmark problems for large-scale continuous optimization. *Information Sciences*, 316:419–436, 2015. 2 citations in sections 2.6.1 and 2.6.4.

[60] Mohammad Nabi Omidvar, Xiaodong Li, Yi Mei, and Xin Yao. Cooperative co-evolution with differential grouping for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 18(3):378–393, 2014. 2 citations in sections 2.8.3 and 2.8.3.

[61] T. Paul and H. Iba. Linear and combinatorial optimizations by estimation of distribution algorithms. In *Proceedings of the 9th MPS Symposium on Evolutionary Computation*, 2002. 2 citations in sections 2.2 and 2.3.1.

[62] Petersen & Pedersen. The matrix cookbook. In *Version: October 3, 2005, Page 53.* 2005. One citation in section 5.1.

[63] D. Peel and G. McLachlan. Robust mixture modelling using the t distribution. *Statistics and Computing, pp. 339-348*, 2000. 3 citations in sections 3.2, 3.2.1, and 4.2.

[64] J.J. Liang K. Deb K. Y.P. Chen A. Auger P.N Suganthan, N. Hansen and S. Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimisation. Technical report, CEC 2005 Special Session on Real-Parameter Optimisation., 2005. 2 citations in sections 5.3 and 5.4.1.

[65] P. Pošík. BBOB-benchmarking a simple Estimation of Distribution Agorithm with Cauchy distribution. In *Proc. of the 11th annual conference companion on Genetic and evolutionary computation conference (GECCO), pp. 2309-2314, ACM*, 2009. 2 citations in sections 3.1 and 4.1.

[66] P. Pošík. Comparison of Cauchy EDA and BIPOP-CMA-ES algorithms on the BBOB noiseless testbed. In *Proc. of the 12th annual conference companion on Genetic and evolutionary computation (GECCO), pp. 1697-1702*, 2010. One citation in section 4.

[67] P. Pošík. Comparison of Cauchy EDA and G3PCX algorithms on the BBOB noiseless testbed. In *Proc. of the 12th annual conference on Genetic and evolutionary computation (GECCO), pp. 1753-1760*, 2010. One citation in section 4.

[68] Mohsen. Pourahmadi. High-dimensional covariance estimation. 2013. One citation in section 2.8.2.

[69] P.Pošík. Preventing premature convergence in a simple eda via global step size setting. In *In the Parallel Problem solving from Nature. Volume 5199 of lecture notes in computer science, pages 549-558.*, 2000. 2 citations in sections 2.7 and 3.1.

[70] R. Raymond and N. Hansen. A simple modification in cma-es achieving linear time and space complexity. In *Parallel Problem Solving from Nature-PPSN . Springer Berlin Heidelberg, 296-305.*, 2008. One citation in section 2.8.

[71] R. Ros and N. Hansen. A simple modification in cma-es achieving linear time and space complexity. In *Proceedings of PPSN, pp. 296-305*, 2008. 2 citations in sections 2.8.4 and 6.5.2.

[72] N.Hansen D. Büche J. Ocenasek S. Kern, S.D. Müller and P. Koumoutsakos. Learning probability distributions in continuous evolutionary algorithms- a comparative review. In *Natural Computing, pp. 77-112.*, volume 3.1, 2004. One citation in section 2.8.4.

[73] M. L. Sanyang and A. Kabán. Multivariate Cauchy EDA Optimisation. In *Proc. of the International Conference on Intelligent Data Engineering and Automated Learning (IDEAL), LNCS 8669, pp. 449-456*, 2014. 5 citations in sections 1.1, 2.3, 2.9, 4.1, and 6.1.

[74] M. L. Sanyang and A. Kabán. Heavy tails with Parameter Adaptation in Random Projection based continuous EDA. In *Proc. of the 2015 IEEE Congress on Evolutionary Computation (CEC), pp. 2074-2081*, 2015. 2 citations in sections 6.1 and 6.5.2.

[75] T. Schaul, T. Glasmachers, and J. Schmidhuber. High dimensions and heavy tails for natural evolution strategies. In *Proc. of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO), pp. 845–852.* ACM, 2011. 7 citations in sections (document), 4, 4.1, 4.3.1, 4.4, 4.4.1, and 4.4.1.

[76] Y.W. Shang and Y.H. Qui. A note on the extended rosenbrock function. *Evolutionary Computation*, 14:119–126, 2006. One citation in section 1.2.

[77] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.P. Chen, A. Auger, and S. Tiwari, editors. *Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real Parameter Optimisation*, 2005. 5 citations in sections 2.6, 3.3, 3.3.1, 4.3, and 4.3.1.

[78] R. Tibshirani T. Hastie and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer-Verlag, 2008. One citation in section 2.3.2.

[79] R. Vershynin. *Introduction to the non-asymptotic analysis of random matrices.* In Y. Eldar and G. Kutyniok, editors, Compressed Sensing, Theory and Applications, Chapter 5, pp. 210-268. Cambridge University Press, 2012. 2 citations in sections 6.6 and 6.6.

[80] Z. Yang W. Chen, T. Weise and K. Tang. Large-scale global optimization using co-operative co-evolution with variable interaction learning. In *Parallel Problem Solving from Nature, Bd. 11, pp. 300-309, 2010.*, 2010. 2 citations in sections 2.8 and 2.8.3.

[81] Y. Wang and B. Li. A restart univariate Estimation of Distribution Algorithm: Sampling under mixed Gaussian and Lévy probability distribution. In *Proc. of IEEE Congress on Evolutionary Computation (CEC), pp. 3917-3924*, 2008. 2 citations in sections 2.9 and 4.1.

[82] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. de Freitas. Bayesian optimization in a billion dimensions via random embeddings. In *IJCAI*, 2013. 13 citations in sections (document), 2.9, 6.1, 6.2, 6.2.1, 6.2, 6.3, 6.3, 6.4, 6.3, 6.4.2, 6.6, and 6.6.

[83] K. Weicker and N. Weicker. On the improvement of co-evolutionary optimizers by learning variable inter-dependencies. In *Proceedings of the Congress on Evolutionary Computation on. Vol. 3.*, 1999. 5 citations in sections 2.3.1, 2.8, 2.8.3, 2.8.3, and 2.9.

[84] Y.Y. Wong, K.H. Lee, K.S. Leung, and C.W. Ho. A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. *Soft Computing. Vol. 7. Issue 8. pp. 506-515*, 2003. One citation in section 5.3.

[85] Q. Xu, M. L. Sanyang, and A. Kabán. Large Scale Continuous EDA Using Mutual Information. In *Proc. of the 2016 IEEE Congress on Evolutionary Computation (CEC)*, 2016 (To Appear). One citation in section 4.2.

[86] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *IEEE Transaction on Evolutionary Computation, Vol. 3, pp. 82-102*, 1999. 6 citations in sections 1.3, 2.7, 2.8.1, 2.9, 3.1, and 4.1.

[87] B. Yuan. and M. Gallagher. On the importance of diversity maintenance in Estimation of Distribution Algorithms. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO), Vol. 1, pp 719-729*, 2005. 3 citations in sections 2.2.2, 2.7, and 3.1.

[88] K. Tang Z. Yang and X. Yao. Multilevel cooperative co-evolution for large scale optimization. In *IEEE World Congress on Computational Intelligence 2008 (CEC 2008)*, 2008. 3 citations in sections 2.8, 2.8.3, and 4.