UNIVERSITY OF AGDER

Gradient descent algorithm incorporating Stochastic Point Location schemes and its application in Multidimensional Scaling analysis

By

Nadja Vidnes

Thesis submitted in Partial Fulfillment of the Requirements for the Degree Master of Science in Information and Communication Technology

Faculty of Engineering and Science

University of Agder

Grimstad

May 2010

Abstract

Gradient descent (GD) is a popular approach for solving optimisation problems. A disadvantage of the method is the difficulties with choosing a proper learning rate that sets the step size for the algorithm. If the learning rate is too small, the convergence is unnecessarily slow, whereas if the learning rate is too large, the process will overshoot or even diverge. Solving optimisation problems that are stochastic in nature introduces additional challenges to the GD algorithm hindering the convergence process.

According to [7], the use of Stochastic Point Location (SPL) schemes can potentially improve any optimisation algorithm by learning the best parameter (the learning rate for GD) during optimisation. In this thesis we have enhanced the classical Gradient descent algorithm by incorporation Linear SPL and Hierarchical SPL schemes. As a result, two new GD based algorithms, GD-LSPL and GD-HSPL, have been developed. The new algorithms iteratively determine the learning rate of the GD in stochastic environments.

To evaluate whether GD-LSPL and GD-HSPL algorithms improve the performance of the GD based optimisation we have compared them with GD with constant learning rate (GD-Classic) and with GD with Line search (GD-LS) algorithms. Three different environments have been used for the empirical evaluation. They are minimising mathematical functions, artificial Multidimensional Scaling (MDS) problems as well as a practical MDS problem concerning the classification of Word-Of-Mouth (WoM) discussions. MDS is a statistical technique used in information visualisation for exploring similarities or dissimilarities in data.

Note that we have proposed and empirically evaluated three different schemes for applying GD-LSPL and GD-HSPL algorithms when identifying the learning rate in the GD based MDS. In brief, the schemes consist of either identifying a global learning rate that is applied to all the points, an unique learning rate for each individual point, or an unique learning rate for every pair of points. These proposed schemes have been rigorously evaluated for artificial MDS problems and for classifying WoM discussions.

The observed results are conclusive — GD-LSPL and GD-HSPL algorithms introduce a clear improvement compared to GD-Classic. Since GD-LSPL and GD-HSPL regulate the learning rate during execution, they achieve faster and more accurate convergence without any manual adjustment of parameters.

While dealing with minimising mathematical functions, we have introduced randomness to the problem data. In these environments GD-HSPL is the fastest and the most accurately converging, the least influenced by variation in the start value and the most frequently converging to the global minimum algorithm. In solving artificial MDS problems the most accurately converging algorithms are GD-LS and GD-HSPL applying learning rate point-wise. These algorithms also perform quite well in Classifying WoM discussions. However, in the latter environment, when the number of points increases, GD-LSPL with the learning rate applied pair-wise has shown the fastest performance.

In solving MDS problems randomness is located in the algorithm itself. GD-LS handles randomness in the algorithm quite well and improvement achieved by GD-HSPL and GD-LSPL is statistically insignificant. However, under certain conditions GD-HSPL and GD-LSPL algorithms have shown potential in improving the performance of GD-LS in solving MDS problems. Moreover, they obviously have shown a great improvement in minimising mathematical functions. We, therefore, believe that GD-HSPL and GD-LSPL algorithms can be utilised in various GD based applications, particularly when randomness of the task is located in the problem data.

Preface

This master thesis is submitted in partial fulfilment of the requirements for the degree Master of Science in Information and Communication Technology at the University of Agder, Faculty of Engineering and Science in Grimstad, Norway. This work was carried out under the supervision of associate professor Ole-Christoffer Granmo at the University of Agder, Norway.

First of all, I wish to thank Ole-Christoffer for assistance and support throughout the project period. He has introduced me to this interesting field of inquiry and has given valuable feedback during the entire project period.

Secondly, I would like to thank my friend Olga Ugland for sharing the experience in writing master thesis, supporting me and helping me with my English skills.

Last but not least I would like to thank my husband Vidar Vidnes and the rest of my family for believing in me and helping me to keep the spirit up. Your support and understanding have made the fulfilment of this study possible.

Grimstad, May 2010.
Nadja Vidnes

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we introduce the Gradient descent (GD) algorithm and its importance in solving optimisation problems. We present challenges of determining the learning rate for the GD, specially in stochastic environments. To meet these challenges, we suggest using Stochastic Point Location (SPL) schemes. Section 1.1 includes the background and motivation of the thesis. It introduces the area of optimisation, GD, SPL and an example of the applications where determining the learning rate for the Gradient descent in stochastic environments has a big influence — Multidimensional Scaling (MDS). This section also presents a practical MDS problem concerning the classification of Word-Of-Mouth (WoM) discussions. Section 1.2 presents the thesis definition stating that the goal of the thesis is the development of new methods which hopefully improve performance of the GD in stochastic environments by incorporating the SPL schemes. Section 1.3 presents our hypotheses. In this section we specify our approach to evaluating whether the new methods improve the GD method, namely by evaluating them in a wide range of environments. In section 1.4 we discuss the contribution of the work. This section states the importance of our work, in case of the trueness of the stated hypotheses, in solving optimisation problems in stochastic environments with high performance. We discuss the consequences of our work in the area of MDS, with a particular focus on classifying WoM discussions, and for applications using GD in general. Section 1.6 shows the structure of the remaining chapters of the report which guides the reader through our investigations, from the theoretical background, via proposed solution and experiments, to conclusion.

1.1 Background and motivation

Optimisation is one of the most important areas of modern applied mathematics. The goal of the nonlinear optimisation problems is the minimisation or maximisation of an objective function involving unknown parameters. The eminent mathematician Gilbert Strang argues in [14] that optimisation is one of the three cornerstones of modern applied mathematics.

Optimisation is stochastic as long as it involves random elements, either in the problem data, in the algorithm itself, or in both. Stochastic optimisation problems originate from the huge variety of the fields that expand from engineering and economics to management science and medicine.

Gradient descent (GD) is a popular approach for solving numerical optimisation problems. The GD method has won great popularity because it is simple in implementation and cheap in execution. It is suitable for large models and works for general cases [9].

Despite the fact that GD is a classic algorithm and there exist a lot of algorithms claiming better performance than GD, it is still used in surprisingly many areas of applications. There may be several reasons for that. The second-orders methods, for example Newtons method, look powerful, but are expensive to compute [2]. They are also very sensitive to the initiation of the start point [7]. In addition, most of the second-orders methods do not work in stochastic environments [9]. Some of the more sophisticated algorithms can simply be too difficult to understand and utilise practically. Therefore, GD remains an important area of research.

1.1.1 Gradient descent

Gradient descent is used to find a local minimum of a function $f(\mathbf{x})$ by following the slope of the function. The gradient gives the direction of the fastest increase. Thus is it natural for minimisation algorithm to go in the direction opposite the gradient a certain amount.

To elaborate, basic Gradient descent algorithm starts with some arbitrary chosen point $\mathbf{x}_0$ and compute the gradient vector for that point. The next value $\mathbf{x}_1$ is obtained by moving some distance from $\mathbf{x}_0$ along the negative of the gradient. In general $\mathbf{x}_{k+1}$ is obtained from $\mathbf{x}_k$ by the equation

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma_k \bigtriangledown f(\mathbf{x}_k) \tag{1.1}$$

where $\bigtriangledown f(\mathbf{x}_k)$ is the gradient of the function at $\mathbf{x}_k$ and γ is a positive learning rate that sets the step size. Hopefully, the sequence $(\mathbf{x}_k)$ converges to the desired local minimum.

The crucial question is one of determining the learning rate, which in turn sets the step size used for the minimisation process. If the learning rate is too small the convergence is sluggish. On the other hand, if the learning rate is too large, the system could diverge.

The principle methods for calculation the learning rate are based on Hessian matrix and Newtons algorithm. They are related to the second derivate of the function being optimised [2].

The disadvantages of the latter methods are:

- If the starting point of the algorithm is not well chosen, the scheme can diverge.
- If the second derivative is small, the scheme is ill-defined.
- Such a scheme requires the additional computation involved in evaluation the (matrix of) second derivatives [7].

According to [2], it is usually easier to set a constant learning rate that is smaller than necessary and make a few more iterations because it takes less time than to compute the optimal learning rate at each step by the Hessian matrix in Newtons algorithm.

Another approach trying to find a suitable value for the learning rate is the Line search algorithm. Line search strategy is an approach finding the minimum value of a function in one dimension. One of the most popular Line search methods, Backtracking-Armijo Line search, is presented in [3]. In GD, using Line search algorithm, there is a need to loosely minimise a function determining the learning rate at each iteration of the GD. Introducing the Line search procedure at each iteration may increase the cost of computation [13].

Solving optimisation problems that are stochastic in nature introduces additional challenges to the GD algorithm. Subsequently, this thesis shows that receiving erroneous responses from the environment can result in many unsuccessful attempts to improve the current position, hindering the convergence process. Furthermore, a random environment makes it difficult to converge to the minimum of the function with a high precision.

For GD with constant learning rate there is a need to manually adjust the learning rate to the problem being solved. The performance of this algorithm is dependent on how well the manually selected learning rate suits the optimisation problem.

According to [7] the use of the Stochastic Point Location (SPL) schemes can potentially improve any optimisation algorithm by learning the best parameter to be used during optimisation, specially in stochastic environments. For the GD algorithm the SPL schemes can potentially help in determining the learning rate. This means that when using GD incorporating the SPL schemes, there is no need in manual adjustment of the learning rate. Moreover, the learning rate does not need to be constant, allowing a better adjustment during execution. This is due to the ability of the SPL schemes to identify the best parameter during optimisation by unsupervised learning.

1.1.2 Stochastic Point Location

Stochastic Point Location (SPL) problem considers a learning automaton trying to locate a point on a line when it is interacting with an environment. The environment tells the automaton in which direction it should move. However, information from the environment can be incorrect.

In 1997 B. John Oommen proposed in [7] the pioneering SPL scheme — Linear SPL. This scheme has the following disadvantage. The learning speed of the scheme decreases linearly when its learning resolution increases. For the Linear SPL scheme higher resolution means that the automaton is taking smaller search steps. Smaller steps means a higher accuracy for the result of the search. However it also takes more time to perform the search, as shown in Figure 1.1. The increased search time comes from the fact that if the search resolution is increased 2

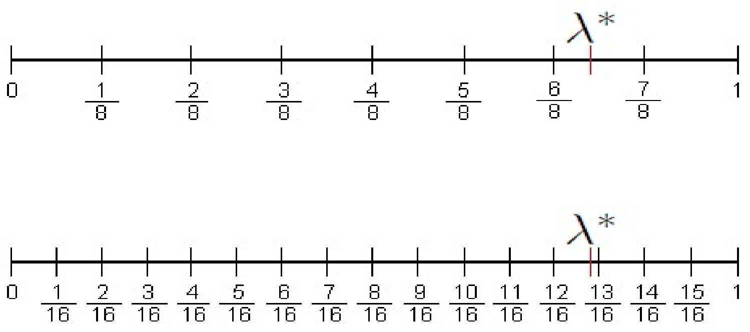

Figure 1.1: Resolution increase causes learning speed decrease for Linear SPL.

times, the length of each search step is cut in half. Therefore, the automaton needs to take 2 times as many search steps resulting in taking up to twice as much time.

It turns out that the above explained conflict between learning speed and learning resolution poses a significant challenge when one needs to solve optimisation problems which require a high solution accuracy.

Ole-Christoffer Granmo has suggested a novel Hierarchical SPL scheme which eliminates the learning speed disadvantage because its searching space is structured as a tree. The Hierarchical SPL scheme has been implemented and evaluated in [1]. Linear and Hierarchical SPL schemes have been compared with respect to resolution and speed. It is shown that the Hierarchical SPL scheme can significantly improve learning speed compared to the Linear SPL scheme. In fact, there is no decrease in learning speed of the Hierarchical SPL when learning accuracy increases. However, for low resolutions Linear SPL scheme achieves a more accurate result and can be preferred. As a result of [1], both Linear and Hierarchical schemes have potential in improving the performance of the GD based optimisation by adaptive determining the learning rate.

Evaluation of the SPL schemes performed in [1] is based on a simulated environment. The schemes have not been applied to any practical optimisation algorithm. In our thesis we apply the SPL schemes to the GD algorithm for determining the leaning rate and evaluate the new algorithms. We are particularly interested in evaluation of the proposed algorithms in solving the optimisation problems that are stochastic in nature. An example of such problem is Multidimensional Scaling.

1.1.3 Multidimensional scaling

Multidimensional Scaling (MDS) emerges from our inability to visualise the structure of multidimensional data when we try to explore similarities or dissimilarities presented in it. In MDS we try to represent data points of multidimensional data (original points) as points in some lower-dimensional space (reproduced points) in such a way that the distances between points in reproduced space correspond to the dissimilarities between points in the original space. If acceptably accurate presentations can be found in two or tree dimensions, this can be an extremely valuable way to gain insight into the structure of the data.

Usually it is not possible to find a configuration when mentioned distances correspond equally well to dissimilarities between all points. That is why we need a criterion (criterion function or loss function) for deciding whether one configuration is better than another. The smaller the value of the criterion function, the better is the fit of the reproduced distances to the original distances. There are several related measures that can be used as criterion functions [2].

When a suitable criterion function is found, it needs to be minimised. For this purpose a GD based algorithm presented in [2] can be used. In this algorithm both starting configuration and points for movement in lower-dimensional (reproduced) space are selected randomly, giving different result each time. Such random selection makes the environment (the minimisation process that can give feedback to the SPL algorithms) stochastic.

The field of MDS is a well established, but the new challenges appear when the data domains become more complex and dynamic [8]. One such challenging area of using MDS is classifying documents, using their syntactic and semantic information and plotting them in a two-dimensional picture, investigated by Zuoyuan Liang and B. John Oommen in [6].

This task is far from simple since the amount of digitally stored text has grown exponentially the last decade. As shown in [6], MDS analysis is one of the important methods to organise various documents into topical clusters and present them on the map display.

Classifying documents is further made use of in research on classifying very noisy Word-Of-Mouth (WoM) discussion set by B. John Oommen, Ole-Christoffer Granmo and Zuoyuan Liang in [8]. WoM collections consisting of on-line discussing fora introduce additional challenges since they are less well organised than digital libraries and digital magazines. Furthermore, they are worse structured and contain more spelling errors, slang and non standard symbols like smileys.

Applying MDS to Classifying WoM may be useful in monitoring ongoing discussion trends and significant events in different business domains.

In [8] on-line discussions concerning mobile phones have been classified. In the latter work a scheme similar to Linear SPL has been applied to the determining gradient descents learning rate at each iteration. The preliminary results of this research give promising indications that we follow up in our work.

1.2 Thesis definition

We formulate the thesis definition as follows.

In this thesis we present two new GD based algorithms, GD-LSPL and GD-HSPL, developed for usage in stochastic environments. For determining the learning rate GD-LSPL and GD-HSPL incorporate Linear SPL and Hierarchical SPL schemes respectively. We empirically evaluate the proposed algorithms by comparing them with each other, with classical Gradient descent (GD-Classic) and with Gradient descent with Line search algorithm (GD-LS). The new methods are evaluated in artificial environments as well as in solving a practical optimisation problem originated from Multidimensional scaling analysis, in particular classification Word-Of-Mouth discussions.

1.3 Hypotheses

In order to evaluate the improvement introduced by the proposed algorithms we formulate the following hypothesis.

GD-LSPL and GD-HSPL improve the performance of the GD based optimisation in stochastic environments

In order to evaluate the algorithms we study how fast and accurate they obtain the optimal solution.

The evaluation includes the environments described in the following sub-hypotheses.

1. **GD-LSPL and GD-HSPL improve the performance of the GD based optimisation (GD-Classic and GD-LS) in minimising mathematical functions**
   We use minimising mathematical functions as an artificial optimisation problem. The algorithms are used to minimise a selection of mathematical functions. Stochastic environments are simulated using normal distribution in order to evaluate robustness of the algorithms to the random feedback.
2. **GD-LSPL and GD-HSPL improve the GD based minimisation of the criterion function in MDS**
   We generate artificial MDS problems by selecting original points appropriate for manual assessment. The points are selected in 3-dimensional space and are intended to be present in a 2-dimensional mapping. The algorithms are used to minimise the criterion function of the generated MDS problems.
3. **GD-LSPL and GD-HSPL improve the GD based MDS method for Classifying Word-Of-Mouth Discussions**
   We apply algorithms to the GD based MDS method for solving a practical MDS problem

concerning the classification of Word-Of-Mouth discussions. The algorithms are used to minimise the criterion function in MDS analysis.

1.4 Contribution

In this thesis we evaluate GD-LSPL and GD-HSPL algorithms incorporating Linear SPL and Hierarchical SPL schemes in comparison to GD with constant learning rate (GD-Classic) and GD with Line search (GD-LS) algorithms. We develop a test bench for comparing the algorithms through experiments on artificial data and in solving practical optimisation problems. Our special interest is directed to the performance of the algorithms in stochastic environments. We show how efficient the new algorithms are compared to the existing algorithms.

While the previous researches on the SPL algorithms, [7] and [1], are based on the simulated "dummy" environments and only mention potential applications of the schemes, this thesis applies the SPL schemes to a classical optimisation algorithm - Gradient Descent. In this way two new algorithms, GD-LSPL and GD-HSPL, are created.

In many optimisation algorithms there is a need to manually adjust the parameters used for optimisation to the problem being solved. For GD-Classic there is a constant learning rate, for GD-LS there is a constant used by the Line Search algorithm. The performance of these algorithms is depending on how well the manually selected parameters suits the optimisation problem. The SPL algorithms, on the other hand, regulate parameters, in particular learning rate for GD during optimisation. This unsupervised learning which eliminates the need of the manual adjustment of the parameters is an important feature of GD-LSPL and GD-HSPL algorithms.

Incorporation of the SPL schemes into GD algorithm introduces a new way of determining the optimal learning rate for GD approach at each iteration. We show whether the algorithms perform this task with low computational cost. We also believe that the proposed algorithms are adaptive for operation in stochastic environments. Thus, if our hypotheses are true, the new algorithms improve the performance of GD-Classic and GD-LS in solving optimisation problems in stochastic environments.

Furthermore, in this thesis we use GD-LSPL and GD-HSPL algorithms for solving a practical optimisation problem, in our case Multidimensional scaling problem (MDS). We show whether GD-LSPL and GD-HSPL algorithms can improve the performance of the GD based MDS analysis, particularly in classifying Word-Of-Mouth (WoM) discussions.

The challenge of the classifying WoM discussions is the growing amount of data. Therefore, it is important to perform analysis in an efficient manner. It is also important to use algorithms that are adaptive to stochastic environments due to the stochastic nature of the problem. If our hypotheses are true, the new algorithms are more efficient than both GD-Classic and GD-LS in performing MDS analysis.

The previous research in classifying WoM discussions, presented in [8], apply Linear SPL-similar scheme to determine gradient descents learning rate for each iteration. We intend to further develop this work by performing a more thorough evaluation. This is achieved by including several algorithms (GD-Classic, GD-LSPL, GD-HSPL and GD-LS) for the purpose of comparison. Further, we propose different ways of applying SPL schemes in performing MDS analysis by gradient descent. Different possibilities of selecting points and applying the criterion function to them are studied as well. Thus we show which way of applying the SPL schemes to GD gives the best performance for MDS.

In general, we believe that if our hypotheses are true, it can have impact on all the optimisation applications using GD algorithm, specially in stochastic environments. In particular, we find the way to solve optimisation problems in stochastic environments with higher performance, giving faster and more accurate result.

1.5 Limitations

The goal for this thesis is to apply the SPL schemes to the Gradient descent (GD) method in order to improve the performance of the GD based optimisation in stochastic environments. We focus on determining the learning rate of GD. As an example of the GD applications, we study GD based MDS analysis where GD is used for minimising criterion function.

Due to the size of the addressed problem area, there is a need to limit the scope of this thesis and make some assumptions.

1. Gradient descent is used as a method for minimising criterion functions. We do not discuss other methods for solving optimisation problems. However, GD is a very much used optimisation algorithm and if we achieve a more efficient algorithm it can have a big impact on many optimisation applications using GD.
2. The focus of the project is MDS. Other practical applications of GD are not studied. However, if we show that the new algorithms can improve the performance of GD in MDS analysis, it arguably means that the same is applicable in general, since we only intend to improve determining the learning rate. The principle of GD is not changed.
3. We select criterion functions for evaluated MDS applications based on previous research. We do not compare different criterion functions for these applications. Obviously, it is important that we use the same criterion function for all compared algorithms. Still, the performance of the algorithms in relation to each other is to a large degree independent of the selected criterion function.
4. We select the way of measuring dissimilarities between objects in evaluated MDS applications based on previous research. We do not compare different ways of measuring dissimilarities. As above, it is important to use the same dissimilarity measure for all the compared algorithms. The performance of the algorithms in relation to each other is to a large degree independent of the selected dissimilarity measure.
5. Since it is known that using Newtons algorithm based on Hessian matrix for determining the learning rate of GD in stochastic environments is not efficient, we do not focus on using this method. What is of interest is to compare the new algorithms with the most efficient existing GD algorithms. That is why we compare them with GD-LS algorithm, in addition to GD-classic.
6. One of the weaknesses of the GD algorithm is that it only finds local minimum, while in many practical problems the interest is directed to the global minimum. Even the main focus of our research is defining the local minimum, it is also of interest to evaluate whether the new methods are better in defining the global minimum compared to the old methods.

1.6 Report outline

The rest of this thesis is organised as follows. Chapter 2 gives theoretical background on previous research in the fields of Gradient descent (GD), Stochastic point location (SPL) and Multidimensional scaling (MDS), including description of the involved algorithms. Here we also present how this knowledge is applied in solving a practical problem, classifying Word-of-Mouth discussions. Chapter 3 presents the proposed solution with description of the new algorithms, GD-LSPL and GD-HSPL, and the ways of applying them in solving MDS problem. Chapter 4 presents results of our experiments evaluating the proposed algorithms, including analysis. Chapter 5 concludes our thesis and points out directions for the further work.

Chapter 2

Background

This chapter presents theoretical background on previous research in the fields of Gradient descent (GD), Stochastic point location (SPL) and Multidimensional scaling (MDS), including description of the involved algorithms. Here we also show how this knowledge is applied in solving a practical optimisation problem, in particular classifying Word-of-Mouth discussions.

2.1 Gradient descent

Gradient descent (GD) is an optimisation algorithm that is used to find a local minimum of a function by iteratively taking steps proportional to the negative of the gradient of the function at the current point.

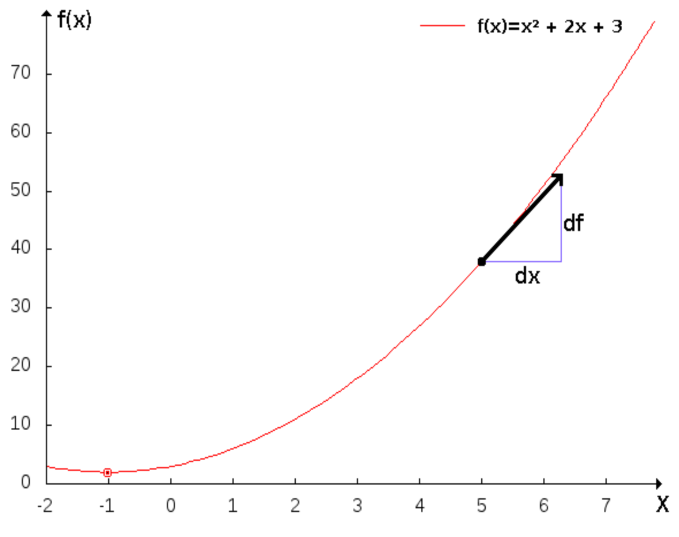

Figure 2.1: Gradient descent for a function in one dimension.

The gradient of a scalar function $f(x_1, x_2, x_3, \ldots, x_n)$ is denoted $\bigtriangledown f(\mathbf{x})$ or $grad\ f(\mathbf{x})$ and is defined to be the vector field whose components are the partial derivatives of $f(\mathbf{x})$, that is

$$\bigtriangledown f(\mathbf{x}) = (\frac{df(\mathbf{x})}{dx_1}, \frac{df(\mathbf{x})}{dx_2}, \ldots, \frac{df(\mathbf{x})}{dx_n}) \tag{2.1}$$

The form of the gradient depends on the coordinate system used. In one dimension the gradient is just the derivative $\bigtriangledown f(x) = \frac{df(x)}{dx}$.

A function is differentiable if the gradient exists [4]. Figure 2.1 shows an example of functions in one dimension $f(x) = x^2 + 2x + 3$. The black arrow represents the gradient of the function $\bigtriangledown f(x) = \frac{df(x)}{dx}$ at point $x = 5$.

The Gradient descent algorithm starts with some arbitrary chosen point $\mathbf{x}_0$ and computes the gradient vector for that point. The next value $\mathbf{x}_1$ is obtained by moving some distance from $\mathbf{x}_0$ along the negative of the gradient. This process is repeated iteratively until length of the gradient vector $||\bigtriangledown f(\mathbf{x})||$ is reduced to the specified value $Tolerance$. The basic GD algorithm is specified below.

```
Algorithm 2.1. Basic Gradient Descent
begin initialise $\mathbf{x}_0$ , threshold $Tolerance, \gamma_0, k = 0$
    do
        $\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma_k \bigtriangledown f(\mathbf{x}_k)$
        $k = k + 1$
    until $||\bigtriangledown f(\mathbf{x}_k)|| \leq Tolerance$
    return $\mathbf{x}_{k+1}$
end
```

where γ_k is a positive scalar factor or learning rate that sets the step size. Hopefully, the sequence $\mathbf{x}_k$ converges to the desired local minimum.

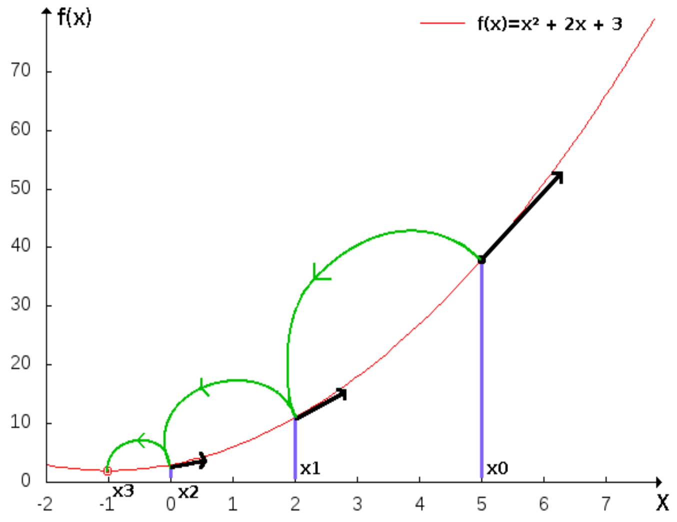

Figure 2.2: Possible steps of the Gradient descent algorithm.

Figure 2.2 presents an example of 3 possible steps of the GD algorithm for function $f(x) = x^2 + 2x + 3$. The search starts at the point $x_0 = 5$, gradient vectors for each step are presented by black arrows and each step is based on the value of the gradient, but taken in the opposite direction. We can see that Gradient descent leads us to the bottom of the function, that is to the point where the value of the function is minimal, $x = -1$.

The value of the learning rate γ_k is allowed to change at every iteration k. However, the choice of the learning rate is a frequently appearing problem. The fast convergence represented in Figure 2.2 is only possible with carefully selected γ_k. If value of the learning rate is too small, the step size will be too small and convergence to the minimum of the function (in our example $x = -1$) is unnecessarily slow, as shown in Figure 2.3. On the other hand, if the learning rate

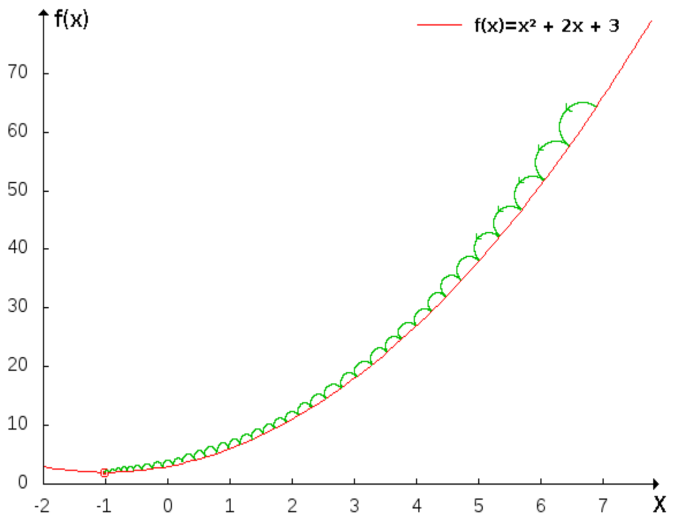

Figure 2.3: Gradient descent with too little step size courses slow convergence.

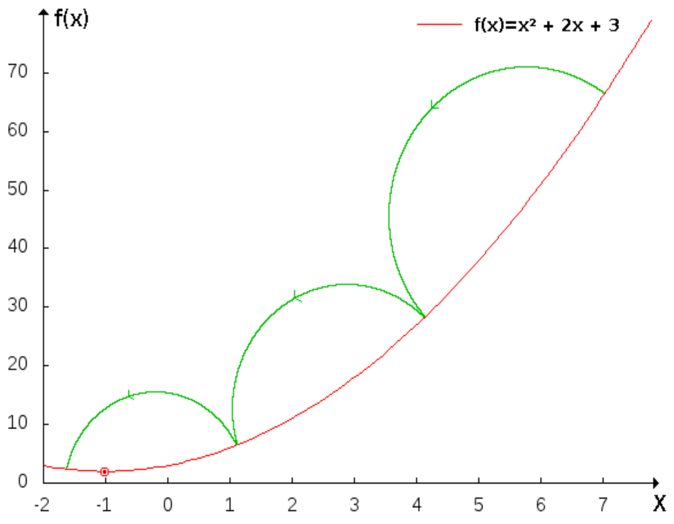

Figure 2.4: Gradient descent with too large step size overshoots.

is too large, the step size will be too large and the searching process can overshoot the desired minimum, as illustrated in Figure 2.4, and can even diverge. In this example GD is not able to converge to the minimum value $x = -1$.

One possible solution to the mentioned problem is to use a constant learning rate that is smaller than necessary and make a few more iterations. This means that γ_k is replaced by a constant γ in Algorithm 1. We refer to this method as Classic GD. The Classic GD method is specified below.

Algorithm 2.2. Classic Gradient Descent

```
begin initialise $\mathbf{x}_0$ , threshold $Tolerance, \gamma, k = 0$
    do
        $\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma \bigtriangledown f(\mathbf{x}_k)$
        $k = k + 1$
    until $|| \bigtriangledown f(\mathbf{x}_k)|| \leq Tolerance$
    return $\mathbf{x}_{k+1}$
end
```

Classic GD does not tell how to select a constant learning rate that both allows convergence (steps are not too large) and, at the same time, lets function converge in acceptable time (steps are not too short). There is a need to adjust the learning rate manually to the problem being solved.

Another approach of finding the learning rate is calculation it at each iteration using Newtons algorithm. This algorithm is based on Hessian matrix, that is matrix of second partial derivatives of the function being optimised. The Newton Descent algorithm is specified below.

Algorithm 2.3. Newton Descent

```
begin initialise $\mathbf{x}_0$ , threshold $Tolerance, k = 0$
    do
        $\mathbf{x}_{k+1} = \mathbf{x}_k - [Hf(\mathbf{x}_k)]^{-1} \bigtriangledown f(\mathbf{x}_k)$
        $k = k + 1$
    until $|| \bigtriangledown f(\mathbf{x}_k)|| \leq Tolerance$
    return $\mathbf{x}_{k+1}$
end
```

where $Hf(\mathbf{x}_k)$ is a Hessian matrix of second partial derivatives $H(f)_{ij}(\mathbf{x}) = (\frac{d^2 f(\mathbf{x})}{dx_i dx_j})$, evaluated at $\mathbf{x}_k$ and $[Hf(\mathbf{x}_k)]^{-1}$ is inverted Hessian matrix.

Newtons algorithm usually gives a greater improvement per step than Classic GD, even with the optimal value of γ_k giving the fastest convergence for Classic GD. However, as mentioned in 1.1.1, Newtons algorithm has several disadvantages. Among other factors is the fact that time order $O(n^3)$ (where n is number of dimensions) is required for matrix inversion at each iteration. This need for time can easily eliminate the improvement achieved by the Newtons algorithm per executed iteration. Therefore, it often takes less time to use Classic GD with constant γ and make a few more iterations than to compute the γ_k at each step using Newtons algorithm. [2]

The third approach of finding the learning rate is to inexactly define a suitable learning rate at each iteration using Line search methods. Line search methods guarantee to pick learning rates that are neither too big nor too small. They loosely minimise $h(\gamma) = f(\mathbf{x}_k - \gamma \bigtriangledown f(\mathbf{x}_k))$ function giving a sufficient decrease in $f(\mathbf{x})$. One of the most used Line search methods is the backtracking-Armijo variant. The Gradient descent with Line search algorithm is specified below.

Algorithm 2.4. Gradient descent with Line search

```
begin initialise $\mathbf{x}_0$ , threshold $Tolerance, k = 0$
    do
        Choose $\gamma_k$ to loosely minimise h($\gamma$) $= f((\mathbf{x}_k - \gamma \bigtriangledown f((\mathbf{x}_k))$ over $\gamma \in \Re^+$
        $\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma_k \bigtriangledown f(\mathbf{x}_k)$
        $k = k + 1$
    until $|| \bigtriangledown f(\mathbf{x}_k)|| \leq Tolerance$
    return $\mathbf{x}_{k+1}$
end
```

The Basic Backtracking Line search algorithm can be used to loosely minimise h(γ) in Gradient descent with Line search. It is iterative algorithm reducing γ by factor τ until decrease in function $f(\mathbf{x})$ is achieved. The Basic Backtracking Line search algorithm is specified below.

Algorithm 2.5. Basic Backtracking Line search

```
begin initialise $\gamma_{init} > 0(e.g.\gamma_{init} = 1), \gamma_0 = \gamma_{init}, l = 0$
    do
        $\gamma_{l+1} = \tau\gamma_l$, where $\tau \in (0, 1)$ (e.g. $\tau = \frac{1}{2}$)
        $l = l + 1$
    until $f(\mathbf{x}_k - \gamma_l \bigtriangledown f(\mathbf{x}_k)) < f(\mathbf{x}_k)$
    $\gamma_k = \gamma_l$
    return $\gamma_k$
end
```

The Basic Backtracking strategy prevents the step size from getting too small since the first learning rate achieving decrease in function $f(\mathbf{x})$ is accepted. However, this method does not prevent too large steps relative to decrease in $f(\mathbf{x})$. This gap is covered by the Armijo condition introducing an additional requirement of sufficient decrease.

The Armijo condition specifies that the learning rate must be selected in a way giving slightly more than simply decrease in $f(\mathbf{x})$. The actual requirement is that

$$f(\mathbf{x}_k - \gamma_l \bigtriangledown f(\mathbf{x}_k)) \leq f(\mathbf{x}_k) + \gamma_l \beta \langle - \bigtriangledown f(\mathbf{x}_k) \bigtriangledown f(\mathbf{x}_k) \rangle; \tag{2.2}$$

for some $\beta \in (0; 1)$ (e.g. $\beta = 0.1$ or $\beta = 0.0001$).

$\langle - \bigtriangledown f(\mathbf{x}_k) \bigtriangledown f(\mathbf{x}_k)\rangle = -\langle \bigtriangledown f(\mathbf{x}_k) \bigtriangledown f(\mathbf{x}_k)\rangle$ is a Euclidean inner product between vectors $- \bigtriangledown f(\mathbf{x}_k)$ and $\bigtriangledown f(\mathbf{x}_k)$ equal to $-\sum_{i=1}^{n} x_i^2$ where n is a number of dimensions.

It can be observed that the larger the learning rate, the larger the required decrease in $f(\mathbf{x})$ [3].

Introducing the Armijo condition into the Basic Backtracking Line search gives the Backtracking-Armijo Line search algorithm specified below.

Algorithm 2.6. Backtracking-Armijo Line search

```
begin initialise $\gamma_{init} > 0$ (e.g. $\gamma_{init} = 1), \gamma_0 = \gamma_{init}, l = 0$
    do
        $\gamma_{l+1} = \tau\gamma_l$, where $\tau \in (0, 1)$ (e.g. $\tau = \frac{1}{2}$)
        $l = l + 1$
    until $f(\mathbf{x}_k - \gamma_l \bigtriangledown f(\mathbf{x}_k)) \leq f(\mathbf{x}_k) + \gamma_l \beta \langle - \bigtriangledown f(\mathbf{x}_k) \bigtriangledown f(\mathbf{x}_k)\rangle$
    $\gamma_k = \gamma_l$
    return $\gamma_k$
end
```

Introducing Line search procedure at each iteration may increase the cost of computation [2].

2.2 Stochastic Point Location

In this thesis we enhance the classical Gradient descent algorithm by incorporation Linear SPL and Hierarchical SPL schemes for determining the learning rate of Gradient descent. The SPL schemes learn the best parameter during optimisation. This unsupervised learning eliminates the need for manual adjustment of constants, in particular learning rate for GD. We evaluate whether these algorithms are efficient and adaptive in stochastic environments.

2.2.1 Problem description

At the core of the Stochastic Point Location (SPL) problem is located a learning automaton (LA) whose task is to determine the optimal value of some variable (or parameter) - λ. We assume that there is an optimal choice for λ - an unknown value, say $\lambda^* \in [0,1]$. The automaton does not know the value of λ^*. Therefore, we study the question of learning this value. We assume that the LA makes some attempts and gets responses from an intelligent environment E which is capable of informing it whether the current value of λ is too small or too big.

The response from the environment is assumed stochastic or "faulty". Thus, E may tell us to increase λ when it should be decreased and vice versa. However, the probability of receiving an intelligent response must be $p > 0.5$. If the environment gives faulty response more than 50% of the time, the LA will not be able to converge on λ^*. The quantity p is also called the "effectiveness" of the environment E. Thus, whenever the current $\lambda < \lambda^*$, the environment correctly suggests that we increase λ with probability p. It simultaneously could have incorrectly recommended that we decrease λ with probability $(1-p)$. Similarly, whenever $\lambda > \lambda^*$, the environment tells us to decrease λ with probability p and to increase it with probability $(1-p)$. [7]

2.2.2 Linear Stochastic Point Location scheme

In 1997 B. John Oommen proposed in [7] the pioneering SPL scheme - Linear SPL that is a way for a learning mechanism to locate a point while interacting with a stochastic environment. The linear SPL scheme suggests how to change our guess of λ^*. This is done in a discretized manner by subdividing the unit interval into N steps $\{0, 1/N, 2/N, ..., (N-1)/N, 1\}$ as shown in Figure 2.5. N is the resolution of the learning scheme. A larger value of N implies a more accurate convergence to the unknown λ^* [7].

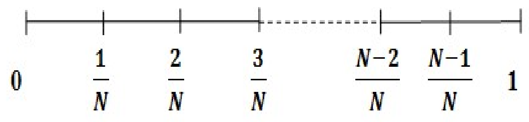

Figure 2.5: The search space of linear SPL is subdivided into N steps.

The Linear SPL scheme which attempts to learn λ^* is specified below. Let $\lambda(n)$ be the value at time step "n". Then,

$\lambda(n+1) := \lambda(n) + 1/N$
  If E suggests increasing λ and $0 \leq \lambda(n) < 1$
$\lambda(n+1) := \lambda(n) - 1/N$
  If E suggests decreasing λ and $0 < \lambda(n) \leq 1$

At the end states the algorithm obeys
$\lambda(n+1) := \lambda(n)$
  If $\lambda(n) = 1$ and E suggests increasing λ
$\lambda(n+1) := \lambda(n)$
  If $\lambda(n) = 0$ and E suggests decreasing λ

Although the above rules are deterministic, because the environment is assumed faulty, the state transitions are stochastic. It is shown in [7] that Linear SPL scheme converges e-optimally and with probability 1.

2.2.3 Hierarchical Stochastic Point Location scheme

As described in 1.1.2, Linear SPL scheme has the following disadvantage. The learning speed of the scheme decreases linearly while its learning resolution increases. Therefore, Ole-Christoffer Granmo has suggested a novel Hierarchical SPL scheme which eliminates the learning speed disadvantage due to its searching space structured as a tree. The automaton searches for the optimal value λ^* by traversing the tree, moving from one tree node to another.

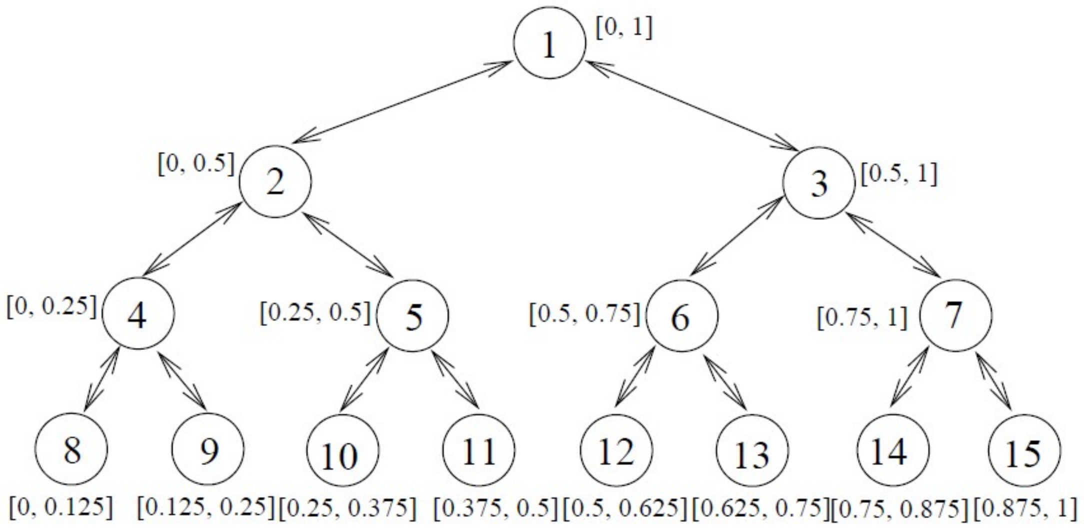

Figure 2.6: The search space of Hierarchical SPL scheme is organised as a tree. Each node of the tree is associated with an interval [a, b].

Structure

As demonstrated in Figure 2.6, each node of the tree is associated with an interval that is a subinterval of [0, 1]. Tree nodes have 2 children, left child and right child. The interval associated with parent node is divided into two equal no-overlapping intervals. They are associated with children nodes. In this way each level of the tree contains twice as many nodes as the previous level. Hence the original interval [0,1] is divided into twice as many subintervals as on the previous level. This means that by increasing the number of levels by 1 we double the resolution of the algorithm.

Thus, in Figure 2.6, the root node is associated with the interval [0, 1], the left child of the root is associated with [0, 0.5], the right child with [0.5, 1]. The resolution of the top level is 1, resolution of the following levels are 2, 4 and 8. Row number 4 contains 8 nodes with intervals [0, 0.125], [0.125, 0.25], [0.25, 0.375], [0.375, 0.5], [0.5, 0.625], [0.625, 0.75], [0.75, 0.875], [0.875, 1]. We can see that resolution is doubled by each level.

Operation

The search starts from the root node. At any iteration the learning automaton finds itself at a current node of the tree. The interval [a, b] associated with this node is sampled at three points. They are leftmost position a, rightmost position b and middle position $(a + b)/2$. For each of these positions environment E may tell us to increase the position (move right = r) or decrease the position (move left = l). The possible responses from the environment received for the three requested positions are {lll,llr, lrl, lrr, rll, rlr, rrl, rrr}. These responses makes the LA move to another node, either to the current nodes parent (Up) or to one of its children (Down Left/Down Right), as indicated by arrows in Figure 2.6.

The possible combinations of responses and movements are summarised in Table 2.1.

a	$(a+b)/2$	b	Move
left	left	left	Up
right	left	left	Down Left
right	right	left	Down Right
right	right	right	Up
left	left	right	Up
left	right	right	Up
left	right	left	Down Right
right	left	right	Down Left

Table 2.1: Possible combinations of responses and movements for Hierarchical SPL scheme

Eventually, based on an informed series of guesses, the LA will be able to converge. It moves within nodes in the tree that are associated with small intervals containing the optimal value λ^*. Convergence of Hierarchical SPL scheme can be proved mathematically by the rule of induction [1].

Algorithm

The Hierarchical SPL scheme returns the middle of the currently selected interval. Resolution N is a number of intervals into which we divide the original interval [0, 1] in the lowest level of the node tree. Resolution requirement implies the number of rows needed in the search tree. For example, in Figure 2.6 there are 8 nodes in the lowest row. This means resolution $N = 8$. To achieve this resolution we need 4 rows. If we say that $\log_2 N = x$, in other words resolution $N = 2^x$, the number of rows in the tree can be calculated as $x + 1$. In our example resolution $N = 8$, then $x = \log_2 N = 3$ and there are $3 + 1 = 4$ rows in the tree.

Each node is characterised by vertical level Row and horizontal position $Column$ as shown in

Figure 2.7. In the start N needs to be initialised and current node is set to be the root node. That means Row and $Column$ are initialised to zero.

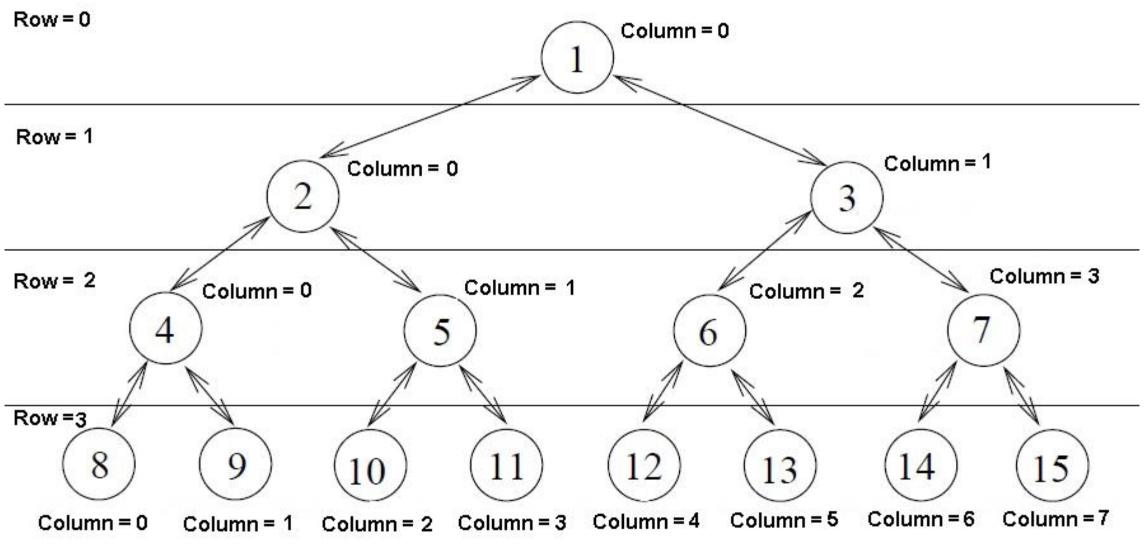

Figure 2.7: Each node is characterised by vertical level Row and horizontal position $Column$

For any current position the LA knows values for Row and $Column$ associated with the node. Based on this knowledge, the LA is able to calculate leftmost, rightmost and middle positions as follows.

```
Leftmost position:
    if $Column = 0$ :
        $left\_most\_position = 0$
    else:
        $left\_most\_position = Column/(2^{Row})$
Rightmost position:
    if $Column = (2^{Row}) - 1$ :
        $right\_most\_position = 1$
    else:
        $right\_most\_position = (1.0 + Column)/(2^{Row})$
Middle position:
    $middle\_position = (1.0 + Column * 2)/(2^{Row+1})$
```

As example, we calculate leftmost, rightmost and middle positions for the node number 5 in Figure 2.7. For this node $Row = 2$ and $Column = 1$.

Leftmost position $= Column/(2^{Row}) = 1/(2^2) = 1/4 = 0.25$.

Rightmost position $= (1.0 + Column)/(2^{Row}) = (1.0 + 1)/(2^2) = 2/4 = 0.5$.

Middle position $= (1.0 + Column * 2)/(2^{Row+1}) = (1.0 + 1 * 2)/(2^{2+1}) = 3/8 = 0.375$

Calculated values are the same as specified in Figure 2.6.

At any current node the LA makes 3 tries (for leftmost, rightmost and middle positions) and receives responses from the environment. The decision about the direction of the movement is done based on the received responses as shown in Table 2.1. The next step is to update the current node. Values for Row and $Column$ are updated based on the selected direction. This is done in the following manner.

```
Down left:
        if $Row < x$
                $Row = Row + 1$
                $Column = Column * 2$
Down right:
        if $Row < x$:
                $Row = Row + 1$
                $Column = Column * 2 + 1$
Up:
        if $Row > 0$:
                $Row = Row - 1$
                $Column = Column/2$
```

Example run

For better understanding of the algorithm we present an example of the execution. Selected resolution is $N = 8$, $x = \log_2 8 = 3$, the search space is as presented in Figures 2.6 and 2.7. Suppose that the destination of the LA is $\lambda^* = 0.4$. This means the destination node is node number 11. LA works as follows.

1. Initial node is a root node with $Row = 0$ and $Column = 0$.
    - LA makes tries at points 0, 0.5 and 1.
    - We assume that correct response is received and it is equal to rll.
    - LA decides movement based on the response and it is Down Left. The decision for the movement is right due to the right response.
    - New current position is calculated. Since $Row < 3$, Row is incremented by 1. New value for Row is 1. $Column$ is multiplied by 2. New value for $Column$ is 0.
2. The current node corresponding to $Row = 1$ and $Column = 0$ is node 2.
    - LA makes tries at points 0, 0.25 and 0.5.
    - We assume that erroneous response is received and it is equal to rlr.
    - LA decides movement based on the response and it is Down Left. The decision for the movement is wrong due to very noisy response.
    - New current position is calculated. Since $Row < 3$, Row is incremented by 1. New value for Row is 2. $Column$ is multiplied by 2. New value for $Column$ is 0.
3. The current node corresponding to $Row = 2$ and $Column = 0$ is node 4.
    - LA makes tries at points 0, 0.125 and 0.25.
    - We assume that erroneous response is received and it is equal to lrr.
    - LA decides movement based on the response and it is Up. The decision for the movement is right because the response is close to the right combination.

- New current position is calculated. Since $Row > 0$, Row is decremented by 1. New value for Row is 1. $Column$ is divided by 2. New value for $Column$ is 0.

4. The current node corresponding to $Row = 1$ and $Column = 0$ is node 2.
   - LA makes tries at points 0, 0.25 and 0.5.
   - We assume that correct response is received and it is equal to rrl.
   - LA decides movement based on the response and it is Down Right. The decision for the movement is right due to the right response.
   - New current position is calculated. Since $Row < 3$, Row is incremented by 1. New value for Row is 2. $Column$ is multiplied by 2 and the result is incremented. New value for $Column$ is 1.
5. The current node corresponding to $Row = 2$ and $Column = 1$ is node 5.
   - LA makes tries at points 0.25, 0.375 and 0.5.
   - We assume that erroneous response is received and it is equal to lrl.
   - LA decides movement based on the response and it is Down Right. The decision for the movement is right because the response is close to the right combination.
   - New current position is calculated. Since $Row < 3$, Row is incremented by 1. New value for Row is 3. $Column$ is multiplied by 2 and the result is incremented. New value for $Column$ is 3.
6. The current node corresponding to the $Row = 3$ and $Column = 3$ is node 11.
7. The destination node is now reached. The final value of the search is equal to the middle position of the destination node (0.375 + 0.5)/2 = 0.4375.

2.3 Multidimensional scaling

We study Multidimensional scaling (MDS) as an example of the optimisation problems that can be solved by the GD method. MDS techniques are used to explore similarities or dissimilarities presented in multidimensional data (original points) by presenting them in some lower-dimensional space (reproduced points). In a similarity matrix a larger score indicates more-similar items whereas in a dissimilarity matrix a larger score indicates less similarity. When objects are equal the similarity is one and the dissimilarity is zero. Thus when objects are very different the similarity is zero and the dissimilarity is one.

The attempt of the MDS analysis is to find such reproduced points that the distances between them correspond to the dissimilarities between points in the original space. If acceptably accurate presentations can be found in two or tree dimensions, this can be an extremely valuable way to gain insight into the structure of the data.

2.3.1 Torgerson Metric model

The simpler MDS model was proposed by Torgerson in [15]. In his model the dissimilarity estimates can be mapped directly to the distances in an Euclidean multidimensional space. For

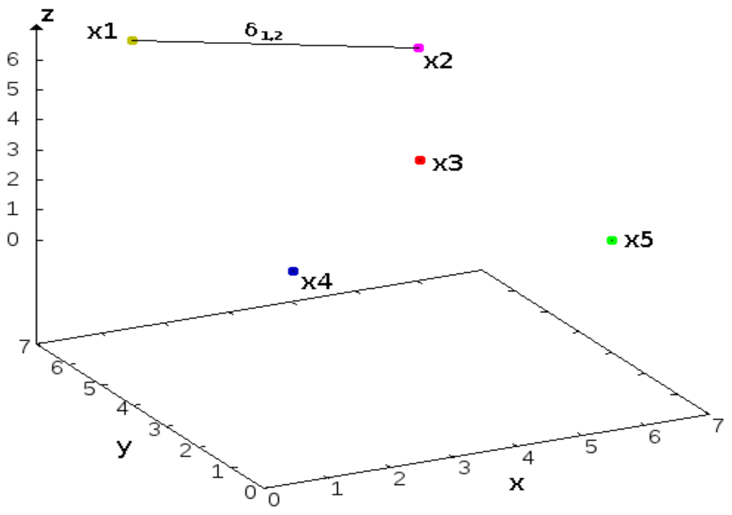

Figure 2.8: Example of points in the 3-dimensional space.

k dimensional space we have:

$$\delta_{ij} = [\sum_k (x_{ik} - x_{jk})^2]^{1/2} \tag{2.3}$$

where δ_{ij} is distance between points x_i and x_j. The matrix of distances $\{\delta_{ij}\}$ is used as input to the MDS analysis.

Below we present example where it is meaningful to talk about distances between points. Figure 2.8 shows points in the 3-dimensional space $x_1, \ldots, x_n$. In our example there are 5 points, meaning than n is equal to 5. Let y_i be the 2-dimensional mapping of x_i, let δ_{ij} be the distance between x_i and x_j and d_{ij} be the distance between y_i and y_j as shown in Figures 2.8 and 2.9.

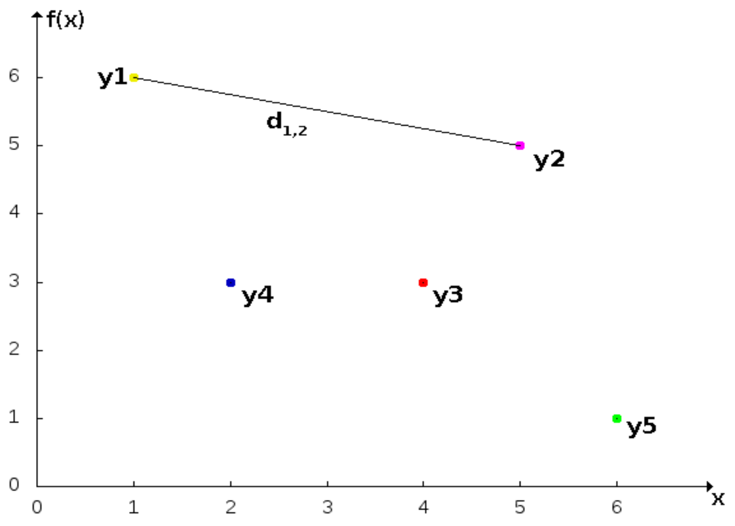

Figure 2.9: Example of 2-dimensional mapping of points.

We are looking for a configuration of points $y_1, \ldots, y_n$ for which the distances $\{d_{ij}\}$ between reproduced points are as close as possible to the corresponding original distances $\{\delta_{ij}\}$. Usually it will not be possible to find a configuration when mentioned distances are equal for all i and j. Thus, a criterion function (criterion) is used to measure whether one configuration is better than another. In brief, the smaller the value of the criterion function, the better is the fit of the reproduced distances to the original distances.

The following sum-of-squared-error function is suggested as an useful criterion since it emphasises large products of error and fractional error [2].

$$S = \frac{1}{\sum_{i<j} \delta_{ij}} \cdot \sum_{i<j} \frac{(d_{ij} - \delta_{ij})^2}{\delta_{ij}} \tag{2.4}$$

An optimal configuration $y_1, \ldots, y_n$ is one that minimise the selected criterion function. According to the MDS procedure described in [2], this can be done by using the GD algorithm starting with random points in low dimensional space. Then at each iteration one of the y_i's, referenced as y_k, is selected. A new position for this point is suggested using calculation of the gradient of the criterion function as presented in equation 1.1. If the new criterion value is less than the old one, the move of the point is done to the new position. The gradient of the above criterion function can be calculated by using equation 2.3 for distances $\{d_{ij}\}$. The result is as follows.

$$\nabla_{y_k} S = \frac{2}{\sum_{i<j} \delta_{ij}} \cdot \sum_{j \neq k} \frac{(d_{kj} - \delta_{kj})}{\delta_{kj}} \cdot \frac{\mathbf{y}_k - \mathbf{y}_j}{d_{kj}} \tag{2.5}$$

Iterations continues until the change in the criterion from one iteration to the next iteration is less than a small number $Tolerance$ such as 0.001 or until a specified maximum number of iterations is reached.

There exist several MDS models developed for different types of MDS applications. Various criterion functions and (dis)similarity measures can be used with different MDS models. Several versions of so-called STRESS functions are often used as measures of fit. They are detailed, among others, in [15], [5] and [6].

The next section presents how MDS analysis is applied in solving a practical problem, in particular classifying Word-of-Mouth discussions.

2.3.2 Classifying Word-Of-Mouth Discussions

Classifying Word-of-Mouth discussions involves several techniques which are explained presently.

Vector Space Model

When classifying documents, each document is represented by Vector space model (VSM) that was first presented by Salton et. al. in [11]. VSM is an algebraic model for representing text documents as vectors of content identifiers called terms. A term can for example be a word or a word phrase extracted from a document collection. An entire document collection called corpus can be viewed as a long string forming a vocabulary of terms.

Each term in a document is assigned a value called weight that represents importance of the term in the document. Thus, a particular document DOC_i is identified by a collection of terms $TERM_{i1}$, $TERM_{i2}$, $\ldots$, $TERM_{it}$ where $TERM_{ij}$ is assumed to represent the weight of term j assigned to document i. $TERM_{ij}$ is set to a positive number when term j actually occurs in document i and to zero when term j is not present in document i.

	$TERM_1$	$TERM_2$	$\dots$	$TERM_t$
DOC_1	$TERM_{11}$	$TERM_{12}$	$\dots$	$TERM_{1t}$
DOC_2	$TERM_{21}$	$TERM_{22}$	$\dots$	$TERM_{2t}$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
DOC_n	$TERM_{n1}$	$TERM_{n2}$	$\dots$	$TERM_{nt}$

Table 2.2: Term assignment matrix with n documents and t terms.

A given document collection may then be represented as a matrix of terms where each row of the matrix represents a document and each column represents the assignment of a specific term to the documents of the collection. A sample term assignment matrix is shown in Table 2.2. [6]

TF-IDF scheme

Salton presents a theory of term importance in automatic text analysis in [12] where he intended to develop a weighting scheme by associating a weight with every token in the document. There he stated that a therm has a great value to a document if it highlight differences or contrasts among other documents in the collection.

Term Frequency-Inverse Document Frequency (TF-IDF) method achieves the goal described in the importance theory. The weighting scheme is achieved by associating weight for each term in a document based on both local information from individual document and global information from the entire corpus of documents [10]. TF-IDF assumes that the importance of a term is proportional to the number of times the term appears in the document, however, offset by the number of documents that it appears in.

Term frequency gives a measure of the importance of the term t_j within the particular document d_i. It is equal to the number of times a given term appears in the document, n_{ij} normalised by the number of occurrences of all terms in document d_i. Thus the Term frequency is defined as follows.

$$tf_{ij} = \frac{n_{ij}}{\sum\limits_k n_{ik}} \tag{2.6}$$

The inverse document frequency is a measure of the general importance of the term. It is obtained by dividing the total number of documents in the corpus, $|D|$ by the number of documents containing the term, $|\{d : t_j \in d\}|$ and then taking the logarithm of the result.

$$idf_j = \log \frac{|D|}{|\{d : t_j \in d\}|} \tag{2.7}$$

It is common to use
$1 + |\{d : t_j \in d\}|$
to avoid division-by-zero in case the term is not in the corpus.

The Term Frequency-Inverse Document Frequency tf-idf is calculated as a product of term frequency and inverse document frequency.

$$(tf - idf)_{ij} = tf_{ij} \cdot idf_j \tag{2.8}$$

Cosine Similarity

Documents presented as vectors of terms using TF-IDF scheme can be compared using Cosine similarity. The latter is a common measure of similarity between two vectors computed as the angle between them. The Cosine similarity between document vectors DOC_i and DOC_j is defined as follows.

$$Sim(DOC_i, DOC_j) = \frac{\sum_{k=1}^{t}(TERM_{ik} \cdot TERM_{jk})}{\sqrt{\sum_{k=1}^{t}(TERM_{ik})^2 \cdot \sum_{k=1}^{t}(TERM_{jk})^2}} \tag{2.9}$$

If the two documents are exactly the same, the corresponding angle between them is zero while the cosine value is 1. [8], [6]

Stress function

Authors of [8] has observed that the task of classifying WoM discussions has the following characteristic feature. It is important to distinguish related topics from unrelated ones in order to present close relationships accurately. Whereas for the non-related topics it is not of significant importance how distant they are. The degree of unrelatedness seems to be more semantic than syntactic. This observation resulted in two following requirements. First of all, objects that are relatively close to each other in the original space should be correspondingly close to each other in the reproduced space. Secondly, objects that are far from each other in the original space should not be placed close to each other in the reproduced space.

A new Stress function (criterion function) has been developed tailored to fulfil the mentioned requirements.

$$S_{WoM} = \sum_{i<j} \frac{(\delta_{ij} - d_{ij})^2}{\delta_{ij} \cdot d_{ij}} \tag{2.10}$$

where δ_{ij} is cosine dissimilarity between DOC_i and DOC_j and d_{ij} is distance between reproduced points for DOC_i and DOC_j.

The stress S_{WoM} measure is doubly-normalised. Thus, the closer the topics, the higher the closeness is prioritised both with respect to the dissimilarity between documents and the distance between reproduced points.

The gradient of the above criterion function can be calculated by using equation 2.3 for distances $\{d_{ij}\}$. The result is as follows.

$$\bigtriangledown_{y_k} S_{WoM} = \sum_{j \neq k} \frac{1}{\delta_{kj} \cdot d_{kj}^2}(2 \cdot (d_{kj} - \delta_{kj}) - \frac{(d_{kj} - \delta_{kj})^2}{d_{kj}}) \cdot (\mathbf{y}_k - \mathbf{y}_j) \tag{2.11}$$

Chapter 3

Proposed solution

This chapter introduces the proposed solution with description of the new algorithms, GD-LSPL and GD-HSPL. Furthermore, we propose how these algorithms can be used to solve MDS problems.

3.1 Linear and Hierarchical SPL schemes

In this project Linear and Hierarchical SPL schemes are used for enhancing the GD algorithm. In other words, we produce new versions of the GD algorithm - GD-LSPL and GD-HSPL. Before we present the new algorithms, let us see how the SPL schemes are supposed to be used in order to determine the value of the learning rate. The Linear and Hierarchical SPL schemes described in 2.2.2 and 2.2.3 respectively can be implemented using the following functions.

Linear SPL

Linear_SPL(resolution):	initiates the scheme and sets the resolution
suggestion():	returns current value of the parameter (in our case the learning rate)
move(direction):	performs the move in the specified direction, Left or Right

In all brevity, we propose the move to the Right (increasing of the learning rate) if the new value of the function, being optimised by GD, is smaller than the previous value. Otherwise, move to the Left (decreasing of the learning rate).

Hierarchical SPL

Hierarchical_SPL(resolution):	initiates the scheme and sets the resolution
left_most():	returns current value of the leftmost position of the current interval (leftmost value of the learning rate)
right_most():	returns current value of the rightmost position of the current interval (rightmost value of the learning rate)
middle():	returns current value of the middle position of the current interval (middle value of the learning rate)
decision (feedback_left_most, feedback_middle, feedback_right_most)	decides the direction of the move in the tree structure: Up, Down Left or Down Right.
move(direction):	performs the movement in the specified direction by updating Row and $Column$ variables

In the case of Hierarchical SPL, we try three different values of the learning rate, namely Leftmost, Rightmost and Middle positions of the current node of the Hierarchical SPL scheme. For each value of the learning rate we propose to move to the right (increase the learning rate) if the new value of the function, being optimised by GD, is smaller than the previous value. Otherwise, move to the left (decrease the learning rate). Thus we collect three feedbacks to the Hierarchical SPL scheme. They are called $feedback\_left\_most$, $feedback\_middle$ and $feedback\_right\_most$ in the algorithm. The direction of the movement in the tree structure of the Hierarchical SPL scheme is decided based on all three feedbacks. Finally, the relocation to the new position is performed.

3.2 GD-LSPL and GD-HSPL algorithms

Combining GD with Linear SPL scheme allows us to find a local minimum of the $f(\mathbf{x})$ while at the same time exploring and utilising a suitable learning rate. We coin this algorithm GD-LSPL and present it below.

Algorithm 3.1. GD-LSPL

```
Input: Start point $\mathbf{x}_0$ , threshold $Tolerance$, Linear SPL scheme resolution $Resolution$
Output: $\mathbf{x}_k$, where $f(\mathbf{x})$ achieves its minimum
$k = 0$
$lspl = Linear\_SPL(Resolution)$
while $|| \bigtriangledown f(\mathbf{x}_k)|| > Tolerance$ do
    $\gamma_k = lspl.suggestion()$
    $\mathbf{x}_{tmp} = \mathbf{x}_k - \gamma_k \bigtriangledown f(\mathbf{x}_k)$
    if $f(\mathbf{x}_{tmp}) < f(\mathbf{x}_k)$ then
        $\mathbf{x}_{k+1} = \mathbf{x}_{tmp}$
        $lspl.move("right")$
    end
    else
        $\mathbf{x}_{k+1} = \mathbf{x}_k$
        $lspl.move("left")$
    end
    $k = k + 1$
end
```

GD-LSPL algorithm extends the Basic GD algorithm by adaptively defining the learning rate γ_k at each iteration using Linear SPL scheme, in particular by taking advantage of the $lspl.suggestion()$ call. Further, a new value of position $\mathbf{x}$ called $\mathbf{x}_{tmp}$ is calculated using γ_k. If the value of the function being minimised $f(\mathbf{x})$ is decreased at position $\mathbf{x}_{tmp}$, it seems reasonable to increase the step size of the search. This is based on the suggestion that the functions shape is not changing and we can continue to follow the same direction of decrease. In this case, we try a larger step to speed up the search. In particular, Linear SPL is updated to increase the learning rate next time using the $lspl.move("right")$ call and the position $\mathbf{x}$ is updated to the value of $\mathbf{x}_{tmp}$. Otherwise, if the value of function $f(\mathbf{x})$ is not decreased at position $\mathbf{x}_{tmp}$, the current step size is too large and we try a smaller learning rate in our seek for convergence. In particular, Linear SPL is updated to decrease the learning rate next time using the $lspl.move("left")$ call and $\mathbf{x}$ is not updated. This procedure is repeated until the length of the gradient vector $|| \bigtriangledown f(\mathbf{x})||$ is reduced to or falls below the specified value $Tolerance$.

Following a similar strategy, by combining GD with Hierarchical SPL scheme, we get GD-HSPL algorithm. Just as GD-LSPL, this algorithm allows us to find a local minimum of $f(\mathbf{x})$ while at the same time a suitable learning rate is explored and utilised in an on-line manner. GD-HSPL algorithm is presented below.

Algorithm 3.2. GD-HSPL

```
Input: Start point $\mathbf{x}_0$ , threshold $Tolerance$, Hierarchical SPL scheme resolution
       $Resolution$
Output: $\mathbf{x}_k$, where $f(\mathbf{x})$ achieves its minimum
$k = 0$
$hspl = Hierarchical\_SPL(Resolution)$
while $|| \bigtriangledown f(\mathbf{x}_k)|| > Tolerance$ do
    $f_{min} = f(\mathbf{x}_k)$
    $\mathbf{x}_{tmp} = \mathbf{x}_k$
    $\gamma_{k\_left\_most} = hspl.left\_most()$
    $\gamma_{k\_right\_most} = hspl.right\_most()$
    $\gamma_{k\_middle} = hspl.middle()$

    $\mathbf{x}_{left\_most} = \mathbf{x}_k - \gamma_{k\_left\_most} \bigtriangledown f(\mathbf{x}_k)$
    if $f(\mathbf{x}_{left\_most}) < f(\mathbf{x}_k)$ then
        $\mathbf{x}_{tmp} = \mathbf{x}_{left\_most}$
        $f_{min} = f(\mathbf{x}_{left\_most})$
        $feedback_{left\_most} = "right"$
    end
    else
        $feedback_{left\_most} = "left"$
    end
    $\mathbf{x}_{middle} = \mathbf{x}_k - \gamma_{k\_middle} \bigtriangledown f(\mathbf{x}_k)$
    if $f(\mathbf{x}_{middle}) < f(\mathbf{x}_k)$ then
        $feedback_{middle} = "right"$
        if $f(\mathbf{x}_{middle}) < f_{min}$ then
            $\mathbf{x}_{tmp} = \mathbf{x}_{middle}$
            $f_{min} = f(\mathbf{x}_{middle})$
        end
    end
    else
        $feedback_{middle} = "left"$
    end
    $\mathbf{x}_{right\_most} = \mathbf{x}_k - \gamma_{k\_right\_most} \bigtriangledown f(\mathbf{x}_k)$
    if $f(\mathbf{x}_{right\_most}) < f(\mathbf{x}_k)$ then
        $feedback_{right\_most} = "right"$
        if $f(\mathbf{x}_{right\_most}) < f_{min}$ then
            $\mathbf{x}_{tmp} = \mathbf{x}_{right\_most}$
            $f_{min} = f(\mathbf{x}_{right\_most})$
        end
    end
    else
        $feedback_{right\_most} = "left"$
    end
    $direction = hspl.decision(feedback_{left\_most}, feedback_{middle}, feedback_{right\_most})$
    $hspl.move(direction)$
    $\mathbf{x}_{k+1} = \mathbf{x}_{tmp}$
    $k = k + 1$
end
```

In the same way as GD-LSPL algorithm, GD-HSPL extends the Basic GD algorithm by adaptively determining the learning rate γ_k at each iteration. But since this algorithm use HSPL scheme, it needs to try three different learning rates to define how to change the learning rate. For that reason the algorithm calculate $\gamma_{k\_left\_most}, \gamma_{k\_middle}$ and $\gamma_{k\_right\_most}$ using $hspl.left\_most(), hspl.middle()$ and $hspl.right\_most()$ respectively. Further, three values of position $\mathbf{x}$ — $\mathbf{x}_{left\_most}, \mathbf{x}_{middle}$ and $\mathbf{x}_{right\_most}$ are calculated using corresponding $\gamma_{k.}$

New value of position $\mathbf{x}$ is selected based on the value of the function $f(\mathbf{x})$ at $\mathbf{x}_{left\_most}, \mathbf{x}_{middle}$ and $\mathbf{x}_{right\_most.}$ The information about whether $f(\mathbf{x})$ is decreased at all three calculated positions is used to define how to change the learning rate. This information is collected in $feedback_{left\_most}$, $feedback_{middle}$ and $feedback_{right\_most.}$ The movement direction for Hierarchical SPL is decided based on all three collected feedbacks. Finally, the relocation to the new position is performed. The new position for Hierarchical SPL implies new values of the learning rate for the next iteration. Again, the latter procedure is repeated until the length of the gradient vector $|| \bigtriangledown f(\mathbf{x})||$ is reduced to or falls below the specified value $Tolerance$.

3.3 Applying GD-LSPL and GD-HSPL algorithms to GD based MDS

In this section we propose three different ways of applying GD-LSPL and GD-HSPL algorithms for determining of the learning rate γ at each iteration of the GD based MDS algorithm.

1. A global learning rate is applied to all the points, meaning that the learning rate is based on the history of all recently moved points, not only on the history of the selected point. The current learning rate is independent of the currently selected point, that is equal for all the points.

   Figure 3.1 shows the example of reproduced points with the learning rate applied globally to the function. In Figure 3.1 y_2 and y_4 are targets for the pink and blue points respectively. By target or target position we mean the position giving the minimal value for the criterion function. Both points move in small steps even though the blue one is placed far from its target and could move with larger steps. This is because of the learning rate is identical for all points — those far from their target position and those close to their target position. If the current movement makes the value of the criterion function smaller, the step size for all points is increased. Whereas if the current movement makes the value of the criterion function larger, the step size is decreased for all points.

2. An unique learning rate is applied for each individual point (point-wise), meaning that the learning rate is based only on the history of the currently selected point. In this way points with badly predicted positions can be moved in larger steps while points that are closer to the target can move in smaller steps as shown in the Figure 3.2

   In Figure 3.2 y_2 and y_4 are targets for the pink and blue points respectively. The blue point is placed far from its target and, accordingly, moves with larger steps than the pink one that is placed close to its target. This is because an unique learning rate is applied for each point. If the current movement makes the value of the criterion function smaller, only the step size for the currently selected point is increased. Whereas if the current movement makes the value of the criterion function larger, the step size is decreased only for this point.

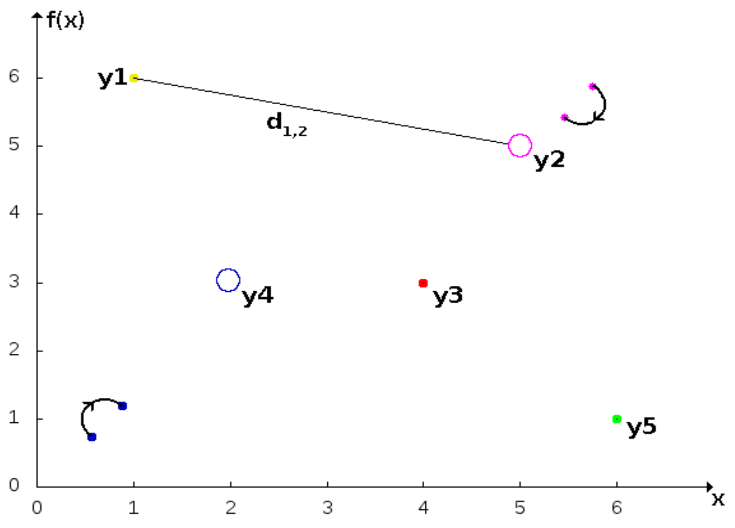

Figure 3.1: Example of 2-dimensional mapping of points. The learning rate is applied globally to the function.

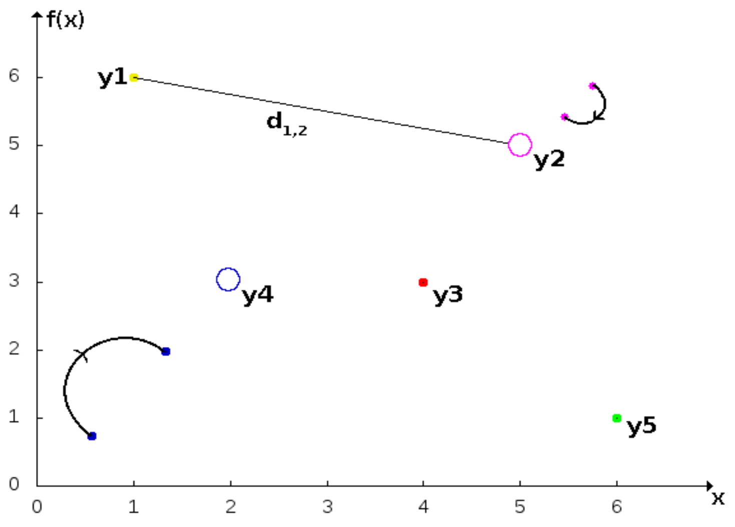

Figure 3.2: Example of 2-dimensional mapping of points. The learning rate is applied point-wise.

3. An unique learning rate is applied for every pair of points (pair-wise). This approach is feasible when we select a pair of points at each iteration instead of a single point. We calculate the criterion function only between those two points as shown in the Figure 3.3

   In our example $d'_{1,2}$ is distance between y_1 and current position of the pink point. y_2 is the target for the pink point and $d_{1,2}$ is distance between y_1 and y_2. The learning rate is adjusted for those two points based on the difference between $d_{1,2}$ and $d'_{1,2}$. We can see that the difference between current distance and target distance is greater for pair y_1, y_4 than for pair y_1, y_2 $((d'_{1,4} - d_{1,4}) > (d'_{1,2} - d_{1,2}))$. For this reason the step size for y_4 is larger than for y_2 in the figure.

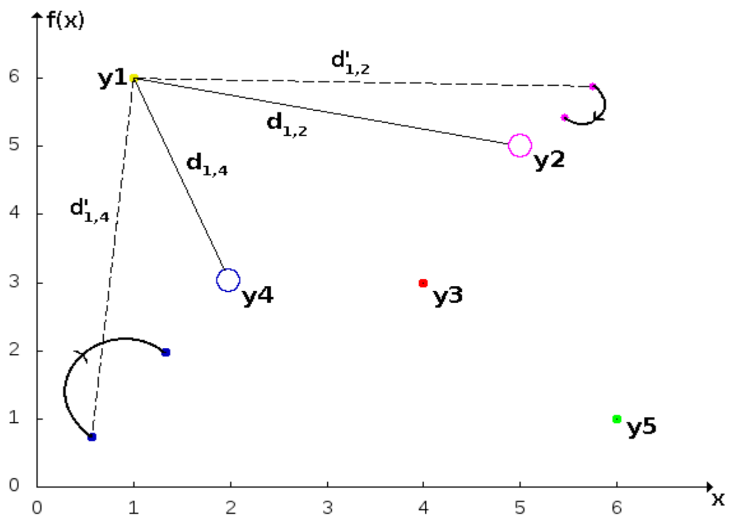

Figure 3.3: Example of 2-dimensional mapping of points. The learning rate is applied pair-wise.

The three mentioned approaches for applying GD-HSPL and GD-LSPL algorithms to the GD based MDS have advantages and disadvantages that we are going to explore. By introducing multiple learning rates, we increase learning flexibility from approach 1 to approach 3. At the same time we reduce number of learning examples applicable to each learning rate. In addition, in approach 3 we use criterion function only between selected points. The advantage of this approach is reduction in calculation time used for each iteration. However, the influence of the current movement on the distances to the other points are not investigated. The possible influence that can be thought is less improvement per iteration.

3.4 GD based MDS with GD-HSPL and GD-LSPL algorithms

Now we present algorithms reflecting ideas introduced in the previous section. All these algorithms have common input and output:

Input: - Dissimilarity matrix of $\{\delta_{ij}\}$ between n objects
- Randomly selected n start points in 2 dimensions associated with (x_i, y_i).
(x_i, y_i) belongs to some 2-dimensional square, e.g. [-2,2]x[-2,2].
- Criterion function, $S(\{(x_i, y_i)\})$
- Linear or Hierarchical SPL scheme resolution, $Resolution$
- Stop criterion
Output: Reproduced points obtained by MDS

Algorithms minimise the specified criterion function iteratively until the given stop criterion is achieved. There are several stop criteria to choose between. Until now we have presented the stop criterion based on the value of the gradient: "Iterations continue until length of the gradient vector $|| \bigtriangledown f(\mathbf{x})||$ is reduced to the specified value $Tolerance$".

Another stop criterion often used in MDS is: "Iterations continue until the change in the criterion from one iteration to the next iteration is less than a small number $Tolerance$".

The third stop criterion could be: "Iterations continue until the maximal number of iterations is achieved or until value of criterion function is reduced to the specified value $Tolerance$".

Algorithm 3.3. GD based MDS with GD-LSPL applied globally to function

```
Initiate global Linear SPL for all the points:
$lspl = Linear\_SPL(Resolution)$
$counter = 0$
while not stop_criterion do
    Select an index $k$ randomly between 1 and $n$.
    Perform GD-LSPL:
    $\gamma = lspl.suggestion()$
    $x_{k\_tmp} = x_k - \gamma \bigtriangledown_{x_k} S$
    $y_{k\_tmp} = y_k - \gamma \bigtriangledown_{y_k} S$
    if $S_{tmp} < S$ then
        $x_k = x_{k\_tmp}$
        $y_k = y_{k\_tmp}$
        $lspl.move("right")$
    end
    else
        $lspl.move("left")$
    end
    $counter = counter + 1$
end
```

In this algorithm we initiate one Linear SPL that is used for all points. GD-LSPL algorithm is used to minimise the criterion function $S(\{(x_i, y_i)\})$. At each iteration one point (x_k, y_k) is randomly selected and gradient is calculated with respect to this point. Further, a new position for the point is calculated. If a smaller value of the criterion function is achieved at the new position, the position of the point is updated. In this case, the Linear SPL scheme is updated to increase the learning rate in the next iteration. If a smaller value of the criterion function is not achieved with the new position, the Linear SPL scheme is updated to decrease the learning rate in the next iteration. In this case, the selected point is left at its current position. Note that the updated learning rate will be used in the next iteration, despite the fact that the selected point will probably be another one.

Algorithm 3.4. GD based MDS with GD-LSPL applied point-wise

```
Initiate an unique Linear SPL for each point:
foreach $(x_i, y_i)$ do
    $lspl_i = Linear\_SPL(Resolution)$
end
$counter = 0$
while not stop_criterion do
    Select an index $k$ randomly between 1 and $n$.
    Perform GD-LSPL:
    $\gamma_k = lspl_k.suggestion()$
    $x_{k\_tmp} = x_k - \gamma_k \bigtriangledown_{x_k} S$
    $y_{k\_tmp} = y_k - \gamma_k \bigtriangledown_{y_k} S$
    if $S_{tmp} < S$ then
        $x_k{=}x_{k\_tmp}$
        $y_k{=}y_{k\_tmp}$
        $lspl_k.move("right")$
    end
    else
        $lspl_k.move("left")$
    end
    $counter = counter + 1$
end
```

In this algorithm we initiate a dedicated Linear SPL for each point. As in algorithm 3.3, GD-LSPL algorithm is used to minimise criterion function $S(\{(x_i, y_i)\})$. At each iteration one point (x_k, y_k) is randomly selected and gradient is calculated with respect to this point. This time the dedicated γ_k for the selected point is used instead of a global learning rate. Further, a new position for the point (x_{k_tmp}, y_{k_tmp}) is calculated. If a smaller value of the criterion function is achieved at the new position, the position of the point is updated. In this case, the Linear SPL scheme for the selected point is updated to increase the learning rate next time the same point is selected. If a smaller value of the criterion function is not achieved at the new position, the Linear SPL scheme for the selected point is updated to decrease the learning rate next time the same point is selected. In this case, the selected point is left at its current position.

Algorithm 3.5. GD based MDS with GD-LSPL applied pair-wise

```
Initiate an unique Linear SPL for each pair of points:
foreach $(x_k, y_k)$, $(x_m, y_m)$ do
  | $lspl_{km} = Linear\_SPL(Resolution)$
end
$counter = 0$
while not stop_criterion do
    Select an index $k$ randomly between 1 and $n$.
    Select an index $m$, different from $k$, randomly between 1 and $n$.
    Perform GD-LSPL:
    $\gamma_{km} = lspl_{km}.suggestion()$
    $x_{k\_tmp} = x_k - \gamma_{km} \bigtriangledown_{x_k} S_{km}$
    $y_{k\_tmp} = y_k - \gamma_{km} \bigtriangledown_{y_k} S_{km}$
    if $S_{tmp_{km}} < S_{km}$ then
        $x_k = x_{k\_tmp}$
        $y_k = y_{k\_tmp}$
        $lspl_{km}.move("right")$
    end
    else
      | $lspl_{km}.move("left")$
    end
    $counter = counter + 1$
end
```

In this algorithm we initiate an a dedicated Linear SPL for each pair of points. GD-LSPL algorithm is used to minimise criterion function $S_{km}((x_k, y_k), (x_m, y_m))$. S_{km} can be found by replacing distances between all points by distances only between (x_k, y_k) and (x_m, y_m) in the criterion function S. For example, for the function in 2.4 S_{km} can be calculated as follows.

$$S_{km} = \frac{1}{\sum_{i<j} \delta_{ij}} \cdot \frac{(d_{km} - \delta_{km})^2}{\delta_{km}} \tag{3.1}$$

At each iteration we randomly select a pair of points (x_k, y_k) and (x_m, y_m). The gradient of S_{km} is calculated with respect to (x_k, y_k). For example, for the function in 2.4 the gradient of S_{km} can be calculated as follows.

$$\bigtriangledown_{y_k} S_{km} = \frac{2}{\sum_{i<j} \delta_{ij}} \cdot \frac{(d_{km} - \delta_{km})}{\delta_{km}} \cdot \frac{\mathbf{y}_k - \mathbf{y}_m}{d_{km}} \tag{3.2}$$

In this algorithm we use γ_{km} for the selected pair. Further, a new position for the point (x_k, y_k), (x_{k_tmp}, y_{k_tmp}) is calculated. If a smaller value of the criterion function S_{km} is achieved at the new position, the position of the point is updated. In this case, the Linear SPL scheme for the selected pair is updated to increase the learning rate next time the same pair is selected. If a smaller value of the criterion function is not achieved at the new position, the Linear SPL scheme for the selected pair is updated to decrease the learning rate next time the same pair is selected. In this case, the selected point is left at its current position.

GD based MDS algorithms with GD-HSPL are produced by the same principle as GD based MDS algorithms with GD-LSPL. The difference is that GD-HSPL algorithm is used and three new points (x_{k_tmp}, y_{k_tmp}) are calculated at each iteration using three values of γ. Since we have already demonstrated how GD-HSPL can replace GD-LSPL when extending Algorithm 3.1 to Algorithm 3.2, we leave that exercise out here for the sake of brevity.

Chapter 4

Results and Analysis

In this chapter we present experiments that have been performed to evaluate the proposed algorithms. We also discuss the outcomes of the experiments. We use three evaluation environments. They are minimising mathematical functions, artificial MDS problems as well as a practical MDS problem concerning the classification of Word-Of-Mouth (WoM) discussions. In all the experiments we compare GD-LSPL and GD-HSPL with each other, with GD-Classic and with GD-LS algorithms.

4.1 Minimising mathematical functions

This section presents experiments where we use minimising mathematical functions in simulated stochastic environments as artificial optimisation problem.

4.1.1 Experimental setup

The experiments are executed until the length of the gradient vector $||\bigtriangledown f(\mathbf{x})||$ is reduced to the specified value $Tolerance$. The lower the length of the gradient vector, the closer the obtained result to the minimum of the function. By defining $Tolerance$ we specify that all algorithms need to achieve the same closeness to the minimum of the function, in other words the same accuracy.

To find out how fast the specified $Tolerance$ is obtained we count the number of responses triggered from the environment until a result within the given $Tolerance$ is achieved. In our case the latter responses correspond to calculation of the function $f(\mathbf{x})$. We refer to this measure as $Number\ of\ iterations$ which forms the y-axis (or z-axis for 3-dimensional plots) of the following plots.

For convenience, we choose to use resolution for SPL schemes, 2^n that is at least as large as the value of $\frac{1}{Tolerance}$. For example, when $Tolerance$ is equal to 0.001 or $\frac{1}{1000}$, we select resolution for SPL schemes to be 2^{10} that is 1024.

There is a need to specify a constant learning rate for GD-Classic algorithm, since it does not adjust the learning rate during execution. For this purpose we try γ equal to $0.1, 0.01, 0.001, ...$ until we found a "standard" value allowing convergence.

For GD-LS we use the Backtracking-Armijo Line search algorithm. There are several parameters to be set for this algorithm. We choose γ_{init} and τ with the intension of GD-LS being similar to GD-LSPL and GD-HSPL. The maximal value of γ is equal to 1.0 for GD-LSPL and GD-HSPL algorithms. For GD-LS the maximal value of the learning rate is equal to γ_{init} which we also set to be 1.0. According to Algorithm 2.6, we multiply γ by τ at each iteration of the Line search algorithm until sufficient reduction in functions value is achieved. For GD-HSPL we divide the searching interval by 2 at each iteration. For the sake of similarity we set τ to be 0.5.

β defines the sufficient decrease in Armijo condition of the GD-LS algorithm. To choose β we try "standard" values like $0.1, 0.01...$ for each experiment and present the best achieved result.

In order to simulate stochastic environments, we introduce normally distributed randomness to the current position (x and y used in calculation of the functions value) of the algorithms. When we wish to increase randomness of an environment, we increase standard deviation σ. Due to randomness of the results, the ensemble average of 100 executions is found in each experiment.

The following functions are minimised in this evaluation.

1. A simple one-dimensional function $f(x) = x^4 + 2x + 3$ with global minimum at $x = -0.79370$, presented in Figure 4.1.

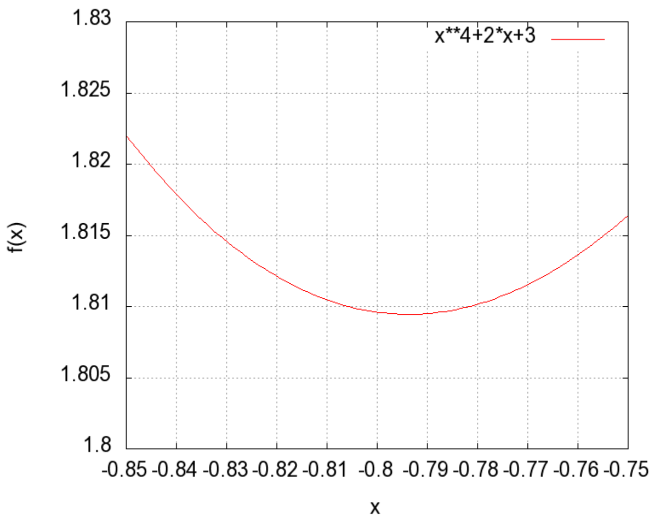

Figure 4.1: Function $f(x) = x^4 + 2x + 3$.

2. A two-dimensional Sinc function $f(x, y) = sinc(x, y) = -(sin(sqrt(x{*}{*}2+y{*}{*}2))/sqrt(x{*}{*}2 + y{*}{*}2))$ with several local minima, presented in Figure 4.2. The global minimum is at $x, y = (0, 0)$.

3. A two-dimensional Six-hump camelback function $f(x, y) = (4 - 2.1 * x^2 + x^4/3) * x^2 + x * y + (-4 + 4 * y^2) * y^2$ with several local minima, presented in Figure 4.3. The two global minima are at $x_1, y_1 = (-0.089, 0.712)$ and $x_2, y_2 = (0.089, -0.712)$.

We have created the following experiments for the evaluation of the algorithms.

Experiment 1. Compare the performance of the algorithms in deterministic environments using varying start values.

Experiment 2. Compare the performance of the algorithms in stochastic environments using varying start values and constant standard deviation for normal distribution.

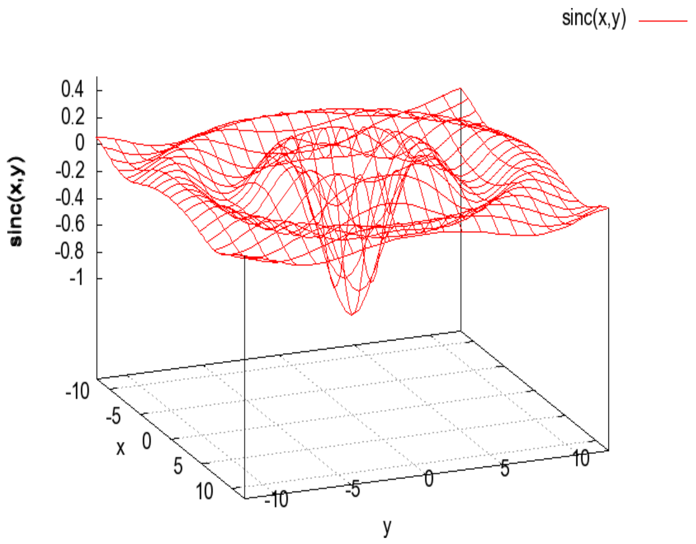

Figure 4.2: Sinc function $sinc(x, y) = -(sin(sqrt(x**2 + y**2))/sqrt(x**2 + y**2))$.

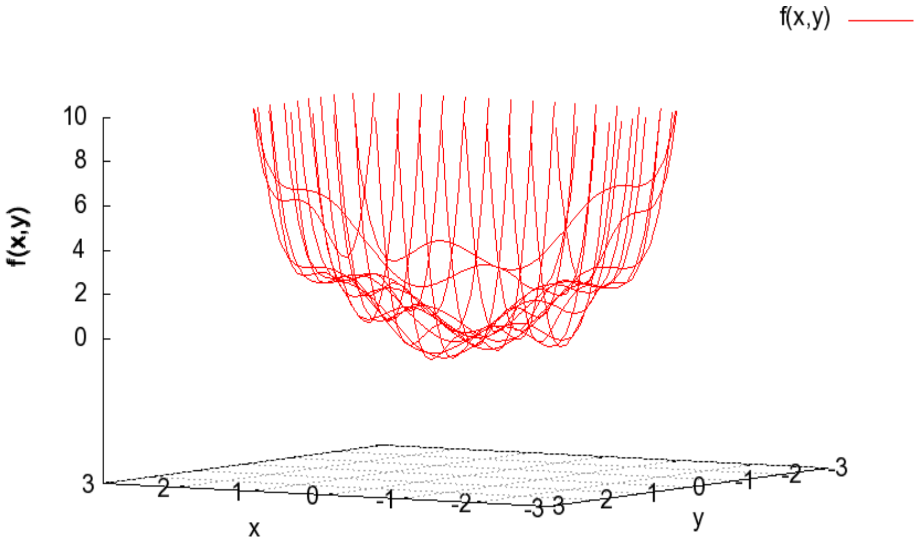

Figure 4.3: Six-hump camelback function $f(x, y) = (4 - 2.1 * x^2 + x^4/3) * x^2 + x * y + (-4 + 4 * y^2) * y^2$.

Experiment 3. Compare the performance of the algorithms in stochastic environments using varying standard deviation for normal distribution and constant start value.

Experiment 4. Compare the performance of the algorithms in stochastic environments using varying $Tolerance$ (accuracy) and constant start value together with constant standard deviation for the normal distribution.

The purpose of the mentioned selection of functions and experiments is to give us a thorough understanding of the behaviour of the algorithms when the start value changes, randomness increases or accuracy increases.

Unless otherwise specified, the following configuration is applied for each of the functions.

Function $\mathbf{f(x) = x^4 + 2x + 3}$

$Tolerance$ is selected to be 0.001. Resolution for SPL schemes is set to be 2^{10}. Learning rate for GD-Classic, γ is selected to be 0.001 in order to achieve convergence for the start values systematically selected from the interval $[-20, -19, \ldots, 19, 20]$. GD-LS achieves the best performance wile using β equal to 0.1.

Sinc function

This function has several local minima. The algorithms converge to 3 different minima while using start values systematically selected from the interval $[1, 2, \ldots, 5] \times [1, 2, \ldots, 5]$. The algorithms converge to the local minima for the most of the start values. We only present results for the start values allowing convergence to the global minimum.

$Tolerance$ is selected to be 0.01. Resolution for SPL schemes is set to be 2^7. Learning rate for GD-Classic γ is selected to be 0.1. GD-LS achieves the best performance without applying the Armijo condition ($\beta = 0$).

Six-hump camelback function

This function has 2 global minima and several local minima. The algorithms converge to 4 different minima while using start values systematically selected from the interval $[0.1, 0, 2, \ldots 3.0] \times [0.1, 0.2, \ldots 3.0]$. For this function too, we present results for the start values allowing convergence to the global minima.

$Tolerance$ is selected to be 0.01. Resolution for SPL schemes is set to be 2^7. Learning rate for GD-Classic γ is selected to be 0.01. GD-LS achieves the best performance without applying the Armijo condition ($\beta = 0$).

4.1.2 Experiment 1

In this experiment we compare the performance of the algorithms in deterministic environments using varying start values. The purpose of the experiment is to identify how fast the algorithms converge to the minima of the functions when environment is deterministic. I addition, we are interested in knowing how the performance of the algorithms is influenced by variation in the start value. A large influence by variation in the start value on the algorithms means a large difference in $Number\ of\ iterations$ achieved by using various start values.

Function $\mathbf{f(x) = x^4 + 2x + 3}$

Results are presented in Figure 4.4.

As can be seen, GD-LS shows the fastest convergence to the minimum of the function being closely followed by GD-HSPL. GD-LS is the least influenced by variation in the start value algorithm. The most influenced by the latter variation is GD-Classic.

Sinc function

Results are presented in Figure 4.5. For this function too, GD-LS is the fastest converging algorithm. It is closely followed by GD-LSPL and GD-HSPL. GD-Classic is the slowest converging algorithm because the learning rate is far from the optimal constant learning rate giving the fastest convergence for GD-Classic. While applying γ equal to 0.5, which is close to the optimal constant learning rate, GD-Classic performs just as fast as the other algorithms.

Algorithms are similarly influenced by variation in the start value.

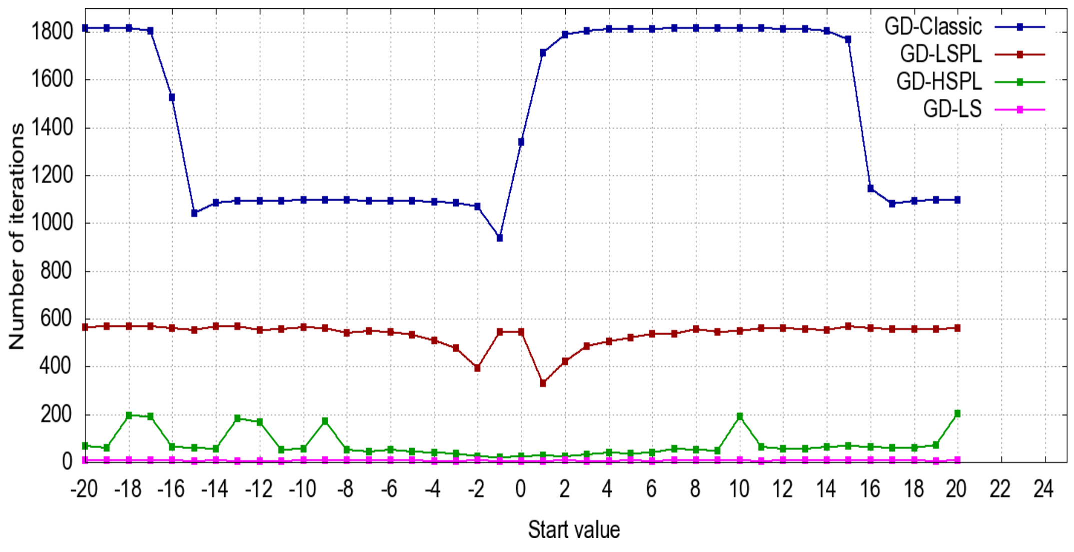

Figure 4.4: $f(x) = x^4 + 2x + 3$. Experiment 1. Deterministic environment.

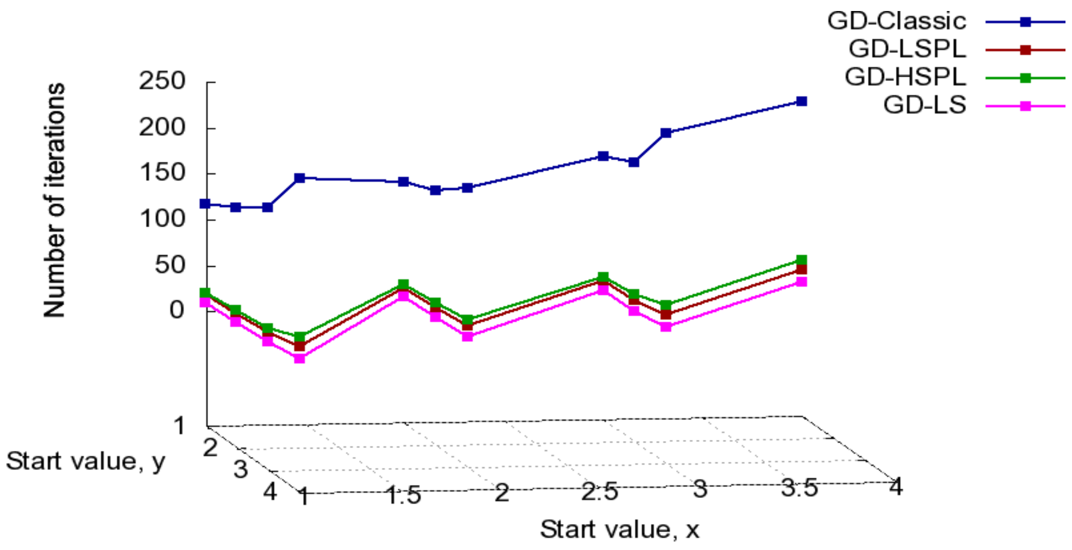

Figure 4.5: Sinc function. Experiment 1. Deterministic environment.

Six-hump camelback function

Results are presented in Figure 4.6. Again, GD-LS shows the fastest convergence being closely followed by GD-HSPL. The convergence speed of GD-Classic is not far from the speed of GD-LS and GD-HSPL. GD-LSPL shows the slowest performance. GD-LS is the least influenced by variation in the start value algorithm. The most influenced by the latter variation is GD-LSPL.

As can be observed, for GD-Classic the least points are plotted, meaning the least frequent convergence to the global minima, in particular 77.8% of the start values. For GD-LS there are the most plotted points, in particular 91.7%, meaning the most frequent global convergence. The percentage of the start values allowing converge to the global minima is 88.9% for GD-LSPL and 83.3% for GD-HSPL.

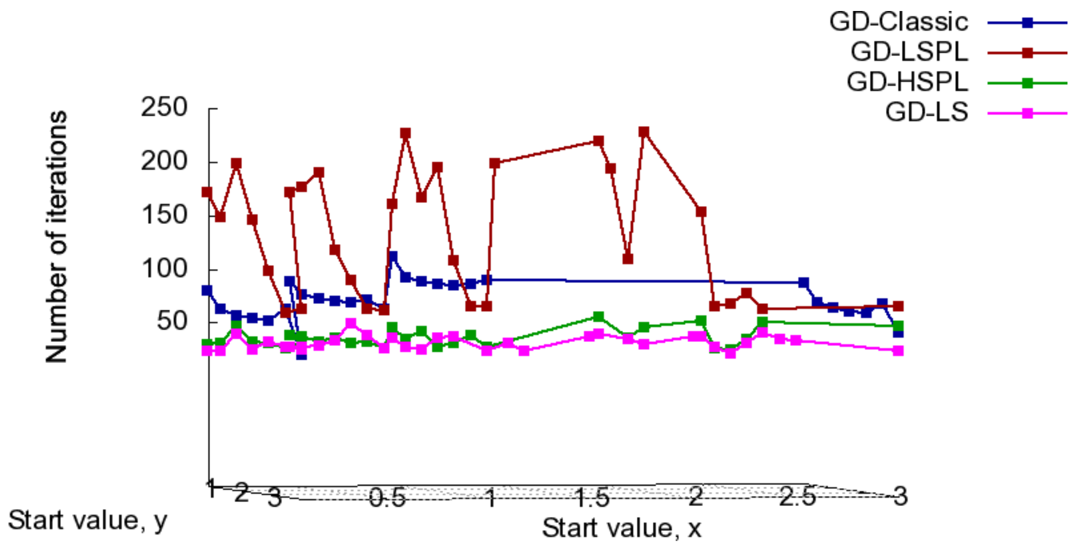

Figure 4.6: Camelback function. Experiment 1. Deterministic environment.

Conclusion for Experiment 1

The reported empirical results show that GD-LS is the fastest performing algorithm in the deterministic environments. GD-LS is also the least influenced by variation in the start value algorithm. In addition, GD-LS shows the most frequent convergence to the global minima. Closely following GD-LS we find GD-HSPL that almost matches GD-LS in performance.

We have seen that performance of GD-Classic is dependent on how close the manually selected learning rate is to the optimal constant learning rate, giving the best performance for GD-Classic. While the learning rate is close to the optimal constant learning rate, GD-Classic converges as fast as GD-LS. Otherwise, the convergence of GD-Classic is slow or fail to appear.

The convergence of GD-LSPL is slower than convergence of GD-LS and GD-HSPL in deterministic environments.

4.1.3 Experiment 2

In this experiment we compare the performance of the algorithms in stochastic environments. To simulate stochastic environments values of x and y used in calculation of the functions value are normally distributed. The standard deviation for the normal distribution σ is constant. The purpose of the experiment is to identify how fast the algorithms converge to the minima of the functions when environment is stochastic. Also for this experiment we are interested in knowing how the performance is influenced by variation in the start value.

Function $\mathbf{f(x) = x^4 + 2x + 3}$

Start values are systematically selected from the interval $[-10, -9, \ldots 9, 10]$. Standard deviation for the normal distribution is selected to be 0.4. GD-LS achieves the best performance without applying the Armijo condition. Results are presented in Figure 4.7.

As can be seen, GD-HSPL is the best performing algorithm. It is both the fastest converging and the least influenced by variation in the start value. GD-Classic and GD-LSPL perform similarly. They converge 6 — 8 times slower than GD-HSPL. GD-LSPL is a bit more influenced

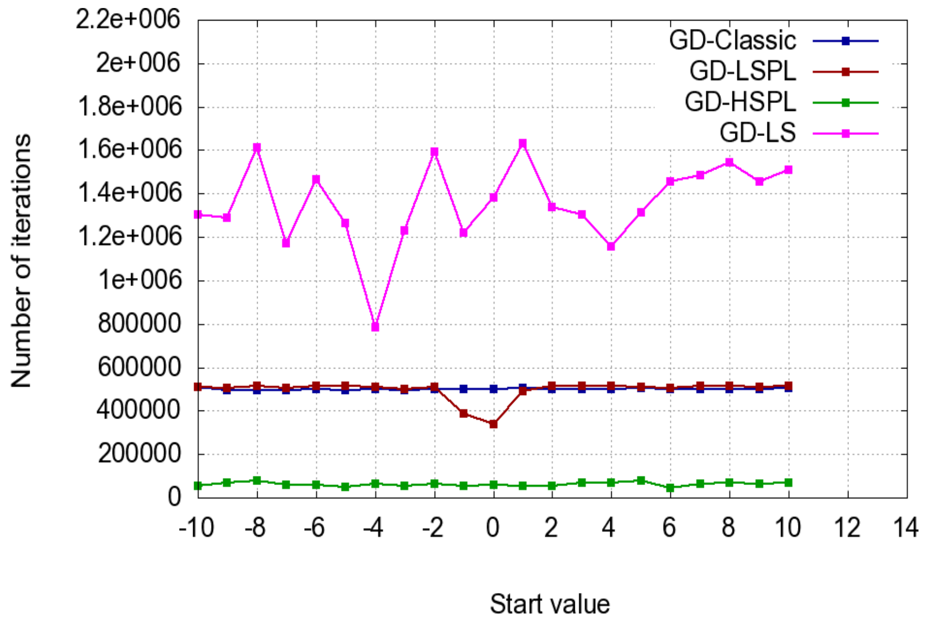

Figure 4.7: $f(x) = x^4 + 2x + 3$. Experiment 2. Normal distributed x, varying start values.

by variation in the start value than GD-Classic. GD-LS is obviously the slowest converging and the most influenced by variation in the start value algorithm.

Sinc function

Start values are systematically selected from the interval $[1, 2, \ldots 5] \times [1, 2, \ldots 5]$. Standard deviation for the normal distribution is selected to be 0.4. Results are presented in Figure 4.8.

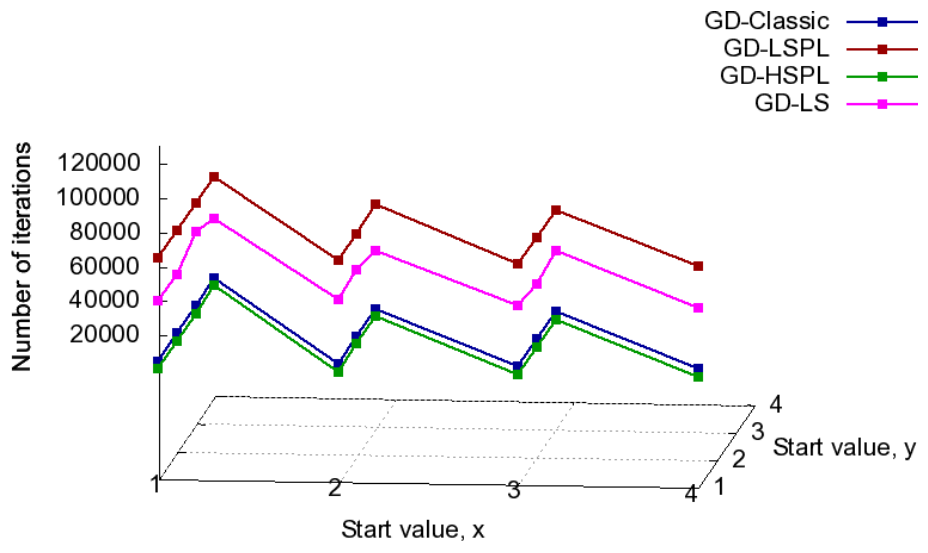

Figure 4.8: Sinc function. Experiment 2. Normal distributed x and y, varying start values.

As can be seen, GD-HSPL is the best performing algorithm also for this function. GD-Classic converges 5 times slower than GD-HSPL. GD-LS and GD-LSPL converges respectively 40 and 65 times slower than GD-HSPL. Algorithms are similarly influenced by variation in the start value.

Six-hump camelback function

Start values are systematically selected from the interval [1,2,. . . 5]x[1,2,. . . 5]. Standard deviation for the normal distribution is selected to be 0.05. Results are presented in Figure 4.9.

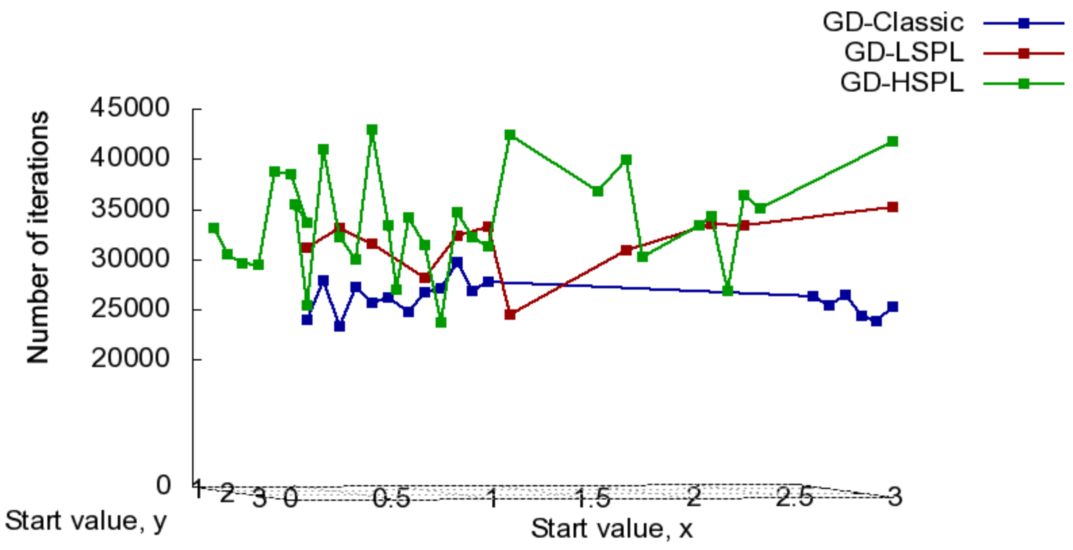

Figure 4.9: Camelback function. Experiment 2. Normal distributed x and y, varying start values.

We can see that performance of GD-Classic, GD-LSPL and GD-HSPL are quite similar. However, GD-Classic is the fastest converging algorithm this time.

GD-LS shows the slowest performance. Its result is not plotted in the figure because it is off scale compared to the other results ($Number\ of\ iterations$ is in range 1-2 millions). In addition, for GD-LS there are the least start values allowing convergence to the global minima, in particular $13,9\%$.

For GD-HSPL there are the most start values allowing convergence to the global minima, in particular 86.1%, meaning the most frequent global convergence. The percentage of the start values allowing convergence to the global minima is 30.5% for GD-LSPL and 50% for GD-Classic.

We can observe that GD-HSPL is more influenced by variation in the start value compared to GD-Classic and GD-LSPL. This observation can be explained by the fact that functions minima lie close to each other. GD-HSPL tries to increase the learning rate in too large steps resulting in jumping around between the functions minima. To slow down GD-HSPL we ignore 90% of the responses giving possible increasing of the learning rate. Results are presented in Figure 4.10. As can be seen, when we ignore the most of the attempts to increase the learning rate, the performance of GD-HSPL is improved and it becomes the fastest converging algorithm.

Conclusion for Experiment 2

GD-HSPL shows the best performance in stochastic environments. It is the fastest and the least influenced by the start value algorithm. When functions minima are very close to each other, GD-HSPL needs to be slowed down to keep the leading position. GD-HSPL is also the most frequently converging to the global minima algorithm.

We can notice that GD-LS is not adaptive to stochastic environments and performs poor in these environments. It converges slower than the other algorithms. GD-LS also has the lowest percentage of the start values allowing convergence to the global minima.

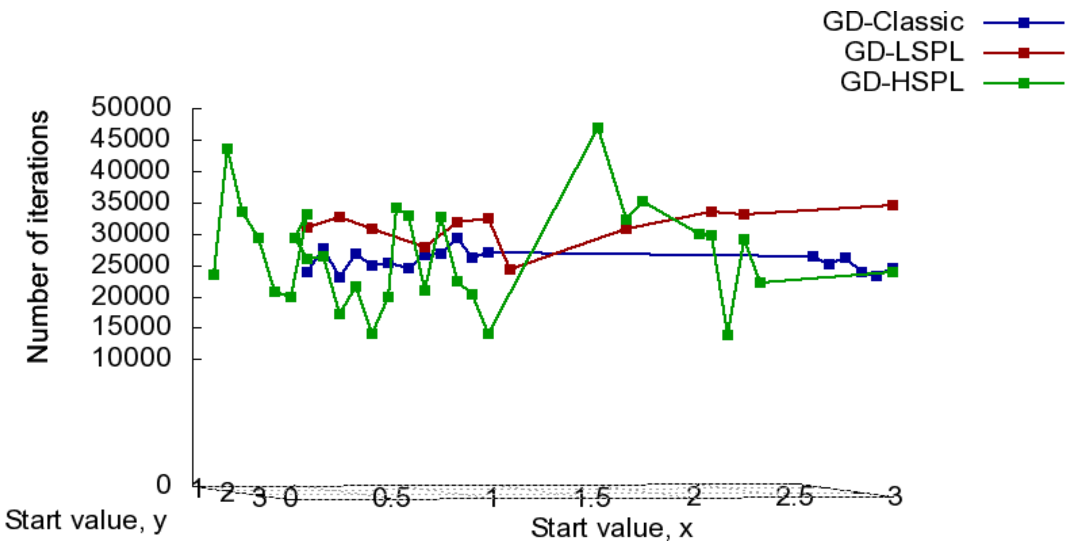

Figure 4.10: Camelback function. Experiment 2. Normal distributed x and y, varying start values. 90% of the positive responses for GD-HSPL are ignored.

GD-Classic performs differently depending on how close the learning rate is to the optimal constant learning rate. GD-LSPL is generally slower than GD-HSPL in stochastic environments.

4.1.4 Experiment 3

The purpose of this experiment is to evaluate how the performance of the algorithms is influenced by increasing randomness. This experiment has been performed with varying standard deviation σ for the normal distribution and constant start value.

Function $\mathbf{f(x) = x^4 + 2x + 3}$

Start value is selected to be 7.0. Standard deviations for the normal distribution are systematically selected from the interval $[0.1, 0.2, \ldots 3.0]$. Results are presented in Figure 4.11.

We can observe that convergence speed of GD-LSPL and GD-classic are similar. Performance of the GD-LS becomes very slow compared to the other algorithms when randomness increases. It can also be seen that GD-HSPL gets greater advantage compared to the competitors when randomness increases. For example, when σ is equal to 1 it takes around 1 million more iterations for GD-LSPL than for GD-HSPL to converge to the minimum of the function. However, when σ is equal to 2 the difference in number iterations between these algorithms becomes close to 2 millions.

Sinc function

Start value is selected to be (3.0,3.0). Standard deviations for the normal distribution are systematically selected from the interval $[0.1, 0.2, \ldots 3.0]$. Results are presented in Figure 4.12.

We can observe that for this function convergence speed of GD-LSPL and GD-LS are similar. GD-Classic performs faster than GD-LSPL and GD-LS. For this function too, when randomness increases, the difference in number iterations between GD-HSPL and the other algorithms increases. Again, GD-HSPL gets the greater advantage compared to the other algorithms when randomness increases.

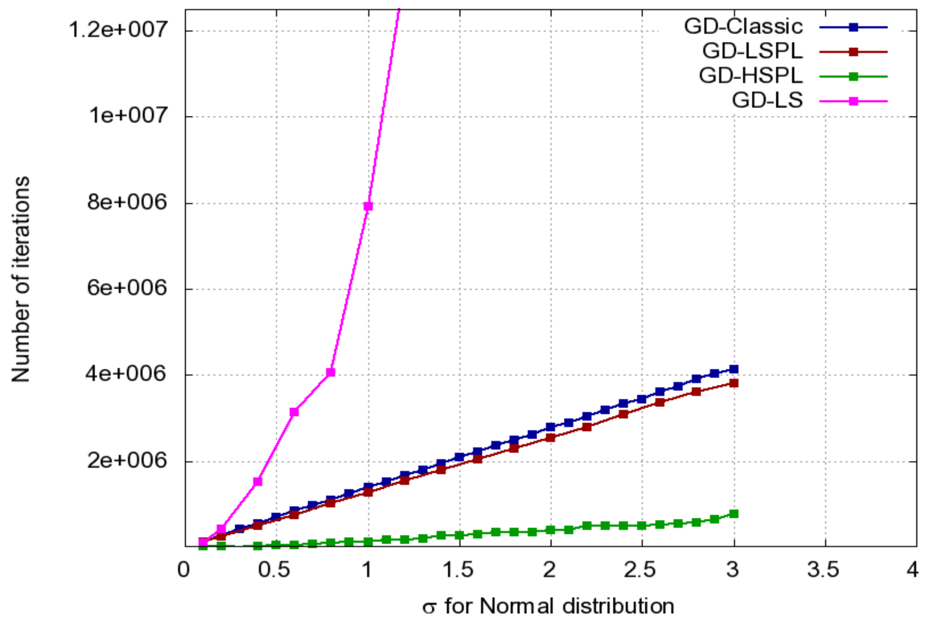

Figure 4.11: $f(x) = x^4 + 2x + 3$, Experiment 3. Normal distributed x, varying σ.

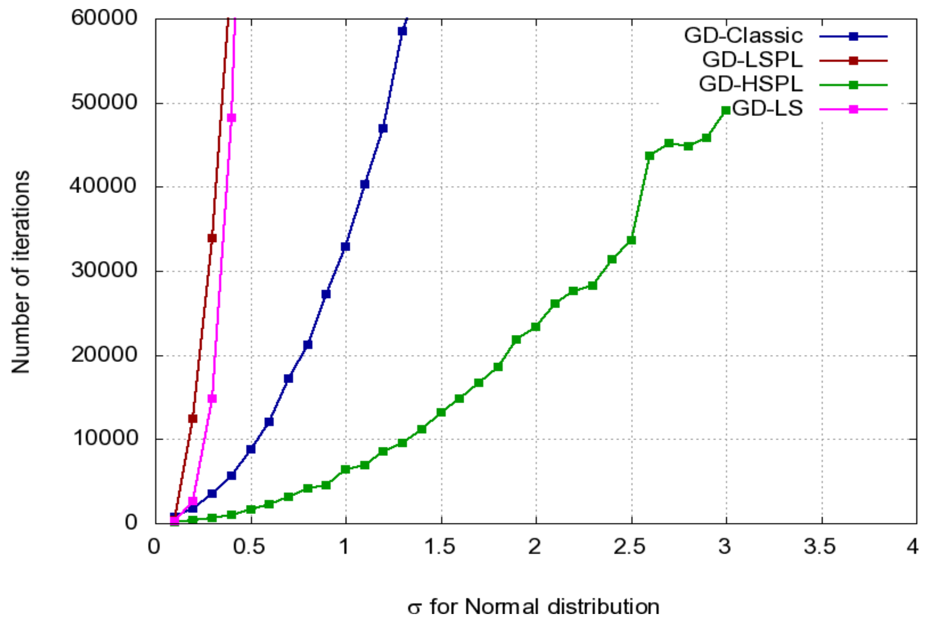

Figure 4.12: Sinc function. Experiment 3. Normal distributed x and y, varying σ.

Six-hump camelback function

Start value is selected to be (1.0,1.0). Standard deviations for the normal distribution are systematically selected from the interval $[0.05, 0.1, \ldots 0.3]$. GD-HSPL is slowed down by ignoring 90% of the responses possibly increasing the learning rate. Results are presented in Figure 4.13.

As can be observed, GD-HSPL is the fastest converging algorithm when randomness is low. However, when randomness increases, GD-HSPL becomes jumping around between local minima. The reason for this is probably the closeness of the functions minima in combination with the large step size of the algorithm. GD-HSPL changes the learning rate in larger steps than e.g. GD-LSPL. Therefore its position changes in larger steps making the movement between

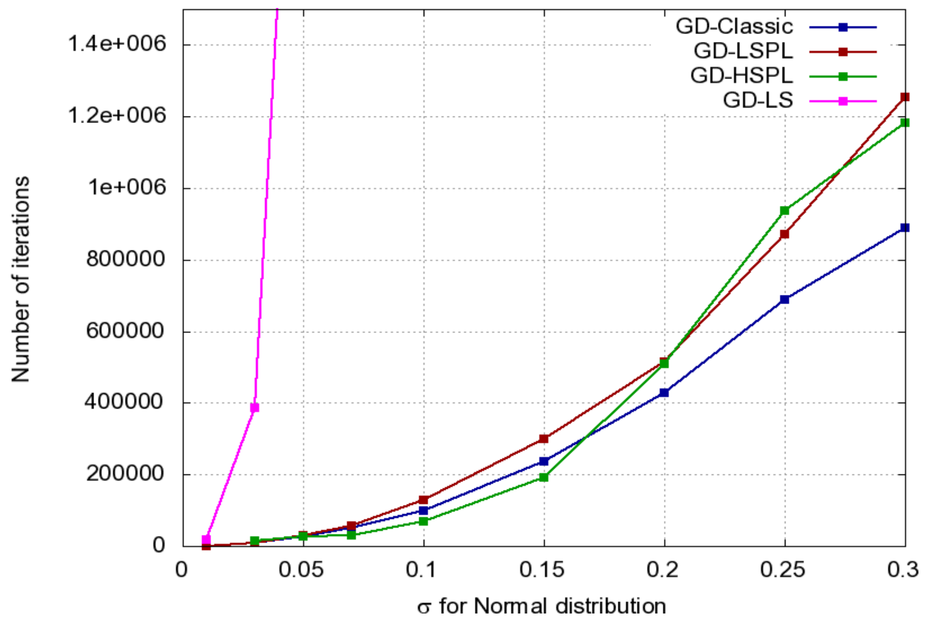

Figure 4.13: Camelback function. Experiment 3. Normal distributed x and y, varying σ. 90% of the positive responses for GD-HSPL are ignored.

functions minima easier. This is particularly noticeable in the presence of high randomness introducing additional mobility. For this kind of functions GD-LSPL becomes a great alternative since it converges as fast as GD-HSPL without being slowed down.

We can also see that GD-Classic shows the fastest convergence for higher values of randomness. The reason for the fast performance of GD-Classic is the closeness of the learning rate γ to the optimal constant learning rate giving the fastest convergence for GD-Classic.

Conclusion for Experiment 3

We can see that GD-HSPL becomes the fastest converging algorithm when randomness increases. The exception from this rule arises in situations when high randomness is combined with very closely located functions minima. In this case GD-LSPL becomes a great alternative.

Performance of GD-LS becomes slow when randomness increases. GD-LS reduces the learning rate each time a new position for x and y does not introduce reduction to the value of the function being minimised. When randomness increases, the number of such unsuccessful attempts rises. Therefore, the learning rate of GD-LS often becomes unnecessarily low giving very slow convergence.

4.1.5 Experiment 4

The purpose of this experiment is to evaluate how the performance of the algorithms is influenced by decreasing $Tolerance$ (increasing accuracy) in stochastic environments. The experiment has been performed with varying $Tolerance$, constant standard deviation σ for the normal distribution together with constant start value. In the figures, representing results for this experiment, we plot variable n increasing along x-axis. $Tolerance$ is defined as 0.1^n and, therefore, decreases along x-axis. Decreasing $Tolerance$ gives increasing accuracy.

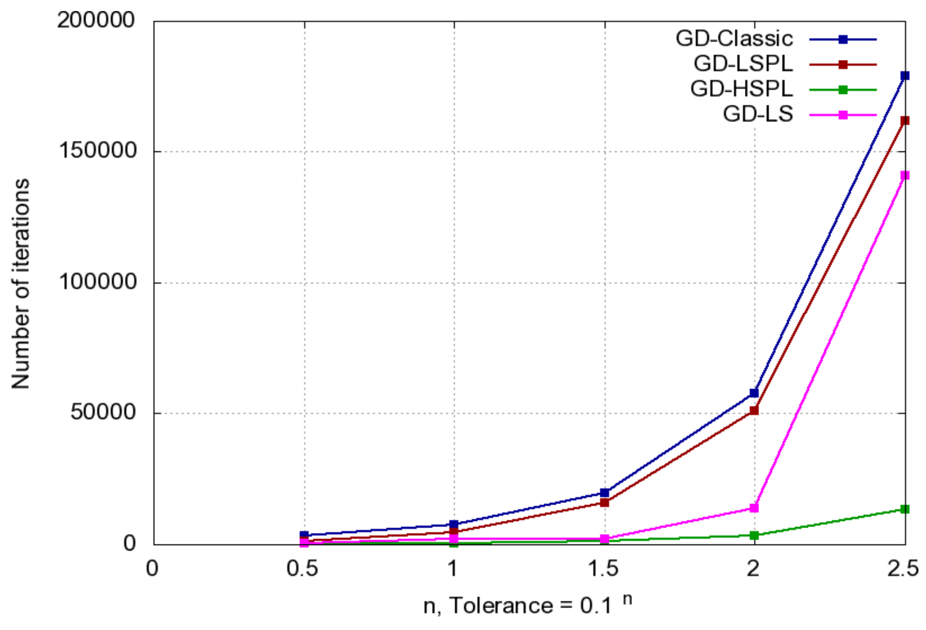

Figure 4.14: $f(x) = x^4 + 2x + 3$, Experiment 4. Normal distributed x, $Tolerance \in [0.1^{2.5}; 0.1^{0.5}]$.

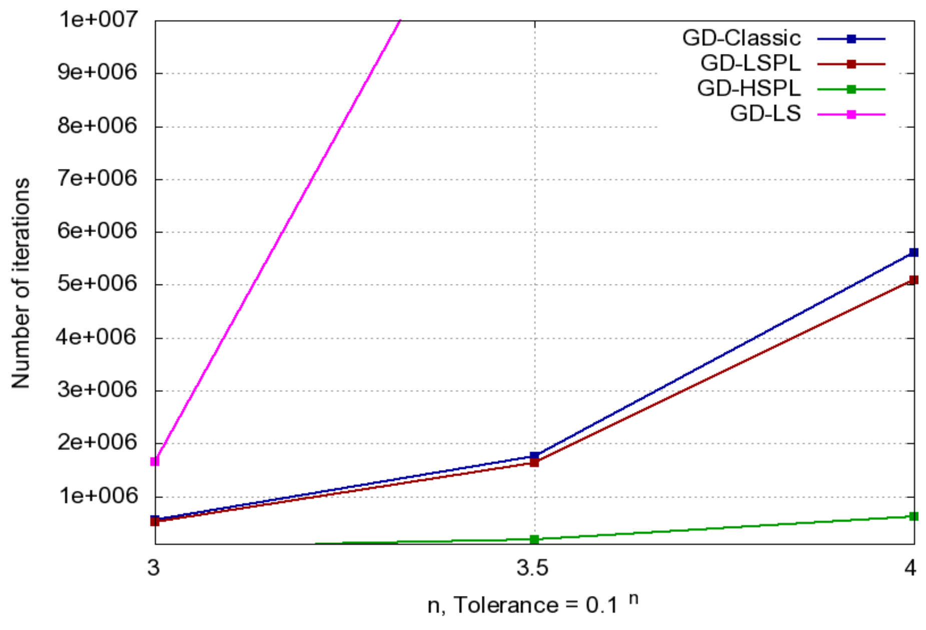

Figure 4.15: $f(x) = x^4 + 2x + 3$, Experiment 4. Normal distributed x, $Tolerance \in [0.1^4; 0.1^3]$.

Function $\mathbf{f(x) = x^4 + 2x + 3}$

Start value is selected to be 7.0. Standard deviation σ for the normal distribution is chosen to be 0.4. $Tolerance$ is systematically selected from the interval $[0.1^4, 0.1^{3.5}, \ldots 0.1^{0.5}]$. GD-LS achieves the best performance without applying the Armijo condition. Results are presented in Figures 4.14 and 4.15.

As can be seen, GD-HSPL is the fastest converging algorithm when $Tolerance$ decreases. GD-LS performs fast when $Tolerance$ is high. However, when $Tolerance$ decreases, the convergence speed of GD-LS becomes unsatisfactory.

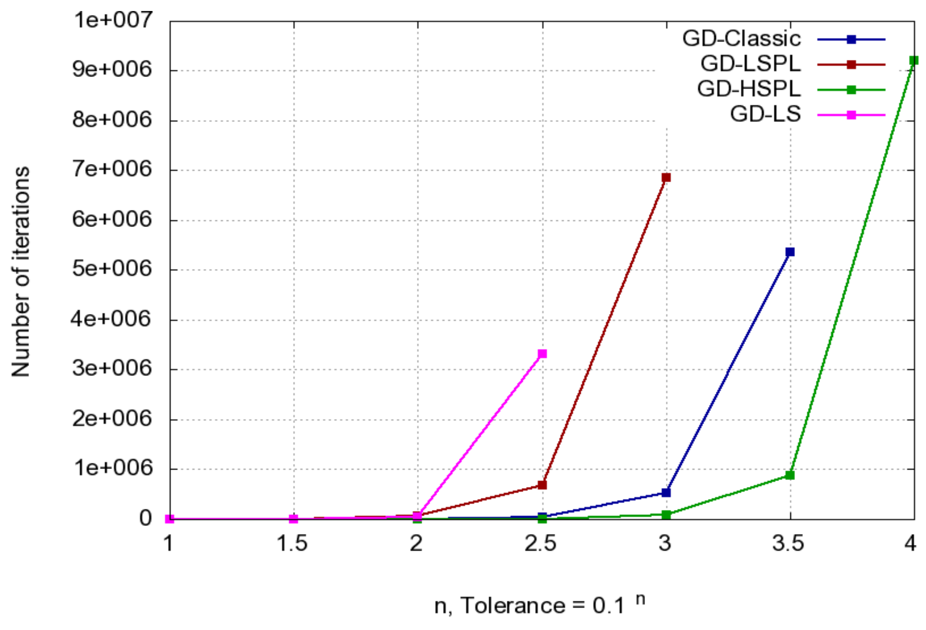

Figure 4.16: Sinc function. Experiment 4. Normal distributed x and y, $Tolerance \in [0.1^4; 0.1^{0.5}]$.

Sinc function

Start value is selected to be $(3.0, 3.0)$. Standard deviation σ for the normal distribution is chosen to be 0.4. $Tolerance$ is systematically selected from the interval $[0.1^4, 0.1^{3.5}, \ldots 0.1^{0.5}]$ Results are presented in Figure 4.16.

As can be observed, GD-HSPL shows the fastest convergence when $Tolerance$ decreases followed by GD-Classic, GD-LSPL and GD-LS. We have also performed the same experiment using $\gamma = 0.5$ for GD-Classic that is close to the optimal constant learning rate. GD-Classic with the optimal constant learning rate performs as well as GD-HSPL when $Tolerance$ decreases.

Six-hump camelback function

Start value is selected to be $(3.0, 3.0)$. Standard deviation σ for the normal distribution is chosen to be 0.05. $Tolerance$ is systematically selected from the interval $[0.1^4, 0.1^{3.5}, \ldots 0.1^{0.5}]$. GD-HSPL is slowed down by ignoring 90% of the responses possibly increasing the learning rate. Results are presented in Figure 4.17.

The result is similar to the result for Experiment 3. For high $Tolerance$ GD-HSPL is the best performing algorithm. However, when $Tolerance$ decreases, GD-HSPL changes the learning rate in too large steps resulting in changing the current position (value of x and y) too much and jumping around the solution.

Conclusion for Experiment 4

We can see that GD-HSPL shows the fastest convergence when $Tolerance$ decreases. The exception from this rule arises in situations when low $Tolerance$ is combined with very closely located functions minima. In this case GD-LSPL becomes a great alternative.

GD-LS performs fast when $Tolerance$ is high. However, when $Tolerance$ decreases, GD-LS becomes the slowest converging algorithm. The reason for slow convergence of GD-LS is probably the same as for Experiment 3, in particular reduction in the learning rate each

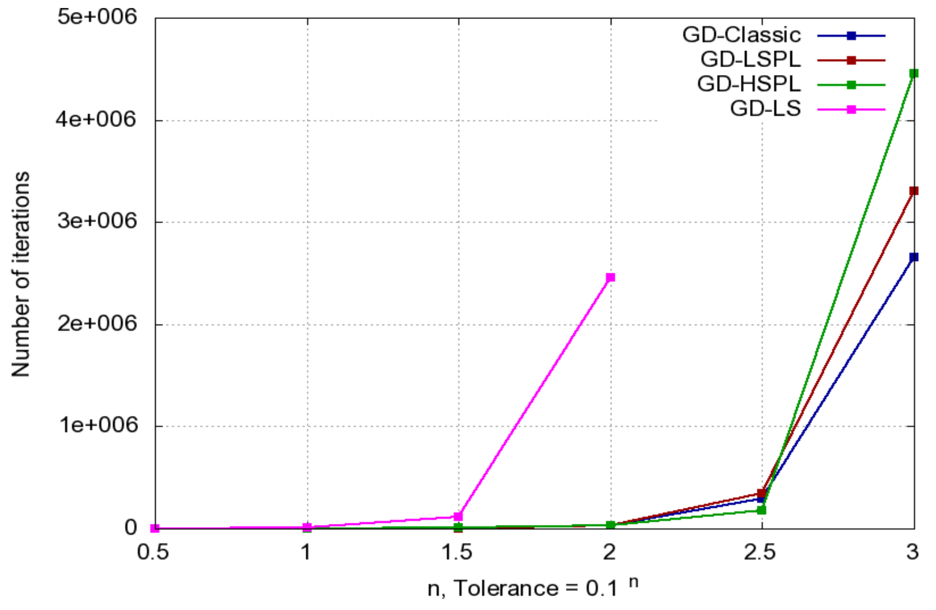

Figure 4.17: Camelback function. Experiment 4. Normal distributed x and y, $Tolerance \in [0.1^3; 0.1^{0.5}]$.

time unsuccessful result is achieved, since number of the unsuccessful attempts increases when $Tolerance$ decreases.

4.1.6 Conclusion for Minimising mathematical functions

We have seen that GD-LS is the best performing algorithm in deterministic environments. It is the fastest converging, the least influenced by variation in the start value and the most frequently converging to the global minima algorithm. However, GD-LS is not adaptive to stochastic environments and converges slow when randomness increases or $Tolerance$ decreases (accuracy increases). In presence of randomness GD-LS is also the most influenced by variation in the start value algorithm.

GD-HSPL becomes the fastest converging and the least influenced by variation in the start value algorithm in stochastic environments. The exception from this rule arises in situations when randomness is high or $Tolerance$ is low and, in addition, local minima are located very close to each other. In this case GD-LSPL becomes a great alternative, even though GD-LSPL is generally slower than GD-HSPL. GD-HSPL also achieves the most frequent convergence to the global minima compared to the other algorithms. Several experiments have shown that speed of the GD-Classic is dependent on the manually adjusted learning rate. GD-Classic with the learning rate close to the optimal constant learning rate converges fast even in presence of high randomness. Even so, GD-HSPL performs as fast as GD-Classic with the optimal constant learning rate, without any adjusted constants.

The reported empirical results are consistent with sub-hypothesis 1 being true. GD-LSPL and GD-HSPL improve the performance of GD based optimisation in minimising mathematical functions in stochastic environments. The latter algorithms are better performing than GD-Classic because they do not need manual adjustment of the learning rate. Compared to the GD-LS they achieve much faster and accurate convergence in stochastic environments. This is specially apparent for GD-HSPL.

4.2 Artificial Multidimensional scaling problems

This section presents experiments evaluating the proposed algorithms applied to GD based minimisation of the criterion function in solving Artificial Multidimensional scaling problems.

4.2.1 Experiment setup

In order to generate MDS problems appropriate for manual assessment we select points in 3-dimensional space that we attempt to present by a 2-dimensional mapping. As a dissimilarity measure between selected points we use distance in Euclidean multidimensional space, as shown in 2.3.

As in the previous section, we count the number of responses triggered from the environment. In this case the latter responses correspond to calculation of the criterion function S defined by 2.4. We refer to this measure as $Number\ of\ iterations$ which forms the x-axis of the following plots.

The experiments are executed until the maximum $Number\ of\ iterations$ that we allow is achieved. Results for each thousandth iteration are reported. The purpose is to compare the performance of the algorithms during operation.

Since we are looking for the minimum value of the criterion function S in all experiments, the most accurate algorithm will be the one achieving the lowest value of the criterion function within the same number of iterations.

Resolution for the SPL schemes is selected to be 2^{10}. For GD-LS we select γ_{init} to be equal to 1.0 and τ to be equal to 0.5 as in the previous section. We also choose β for GD-LS to be 0.001.

The following functions are applied in this evaluation to generate points in 3-dimentional space.

1. Spiral function with 9 original points, presented in figure 4.18.

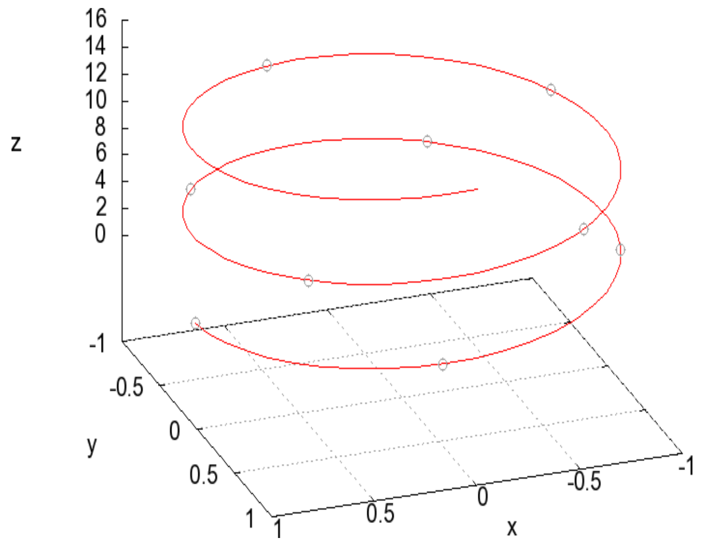

Figure 4.18: Spiral function with 9 original points.

2. Triangle function with 20 original points, presented in figure 4.19.

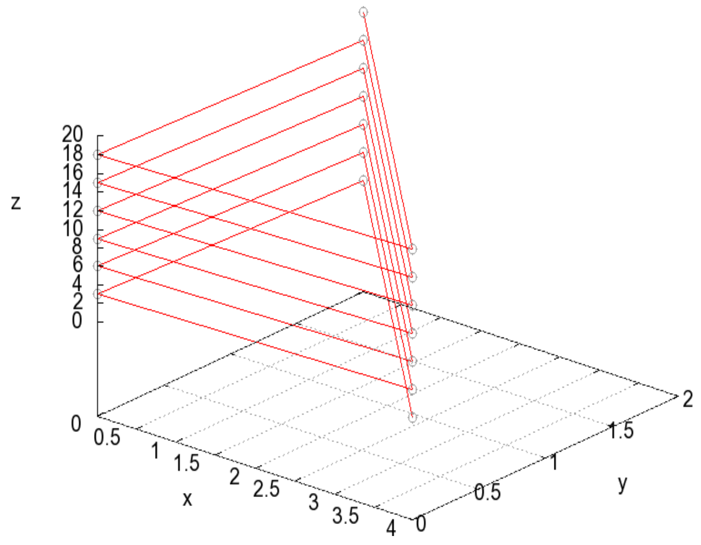

Figure 4.19: Triangle function with 20 original points.

In contrast to what has been discussed earlier, the learning rate for minimising criterion function for the mentioned points is not located between 0 and 1. When bounds of the learning rate LR_{min} and LR_{max} are known, we can still use the λ produced by the SPL schemes. The only thing we need to do is to rescale λ based on the maximum and minimum learning rates as described below.

$$\lambda = \frac{LR - LR_{min}}{LR_{max} - LR_{min}} \tag{4.1}$$

Since LR_{min} is equal to 0, we can calculate learning rate LR as follows.

$$LR = \lambda \cdot LR_{max} \tag{4.2}$$

The maximal learning rate is selected to be 1000.

We have created the following experiments for the evaluation of the algorithms.

Experiment 1. Compare the performance of the GD-HSPL and GD-LSPL algorithms with the learning rate applied to the criterion function globally. Thus, we randomly select one point at each iteration and calculate value of the criterion function S, as described in Algorithm 3.3 for GD-LSPL and correspondently for GD-HSPL.

Experiment 2. Compare the performance of the GD-HSPL and GD-LSPL algorithms with the learning rate applied point-wise. Thus, we randomly select one point at each iteration and calculate value of the criterion function S, as described in Algorithm 3.4 for GD-LSPL and correspondently for GD-HSPL.

Experiment 3. Compare the performance of the GD-HSPL and GD-LSPL algorithms with the learning rate applied to each pair of points (pair-wise). Thus, we randomly select two points at each iteration and calculate value of the criterion function only based on the distance between them, as described in Algorithm 3.5 for GD-LSPL and correspondently for GD-HSPL.

We randomly generate 100 sets of initial values, each point is drown from the interval $[0; 12]$ for the Spiral function and from the interval [0;10] for the Triangle function. The ensemble average of 100 executions (one execution for each set of initial values) is found.

GD-LS and GD-Classic algorithms do not remember the learning rate from the previous iteration. Therefore, the tree ways of applying the learning rate, meaning remembering the learning rate, evaluated by the experiments are not applicable for these algorithms. However, we include the results for GD-LS and GD-Classic for comparison purposes.

4.2.2 Experiment 1

In this experiment we compare the performance of GD-LSPL and GD-HSPL algorithms with the learning rate applied to the criterion function globally as described in 3.3(1).

Spiral function

Results are presented in Figure 4.20.

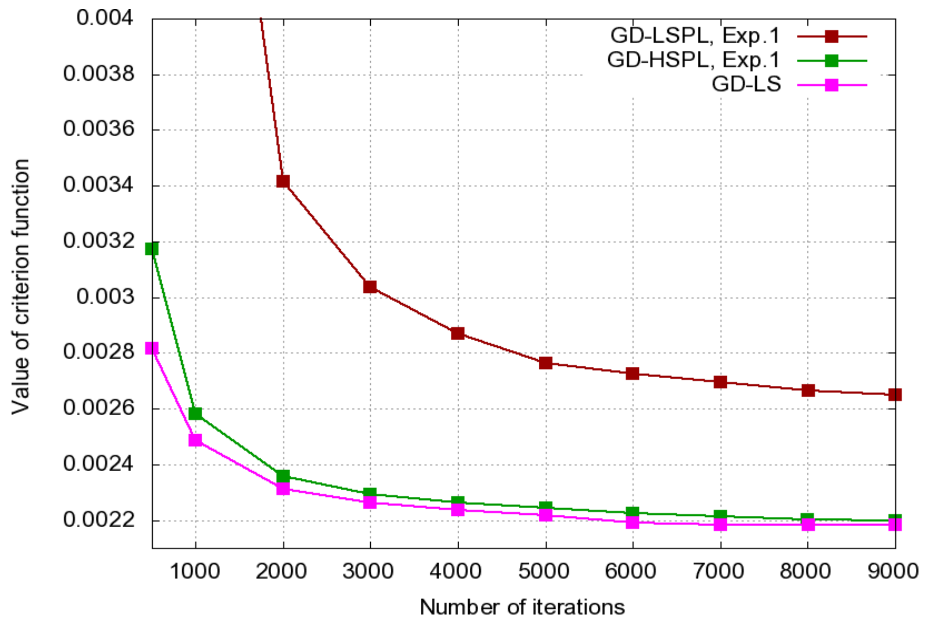

Figure 4.20: Experiment 1. Spiral function. Learning rate is applied to the criterion function globally. Results for GD-LS are plotted for comparison purposes.

As can be seen, GD-HSPL with the learning rate applied globally to the criterion function and GD-LS provide similar performance. Repetition of the experiment has shown that the difference in converging speed and accuracy between GD-LS and GD-HSPL is statistically insignificant. In addition, GD-LS may be a bit influenced by the value of β and, therefore, its position may vary a bit from the shown position.

GD-LSPL converges slower and less accurately than GD-LS and GD-HSPL. Result for GD-Classic is not included because the algorithm does not manage to converge with the learning rate equal to 1000. Performance of GD-Classic with the learning rate close to the optimal constant learning rate equal to 100 is similar to performance of GD-LS and GD-HSPL.

The best value of the criterion function achieved during this experiment was 0.00186. Reproduced points for this result are presented in Figure 4.21.

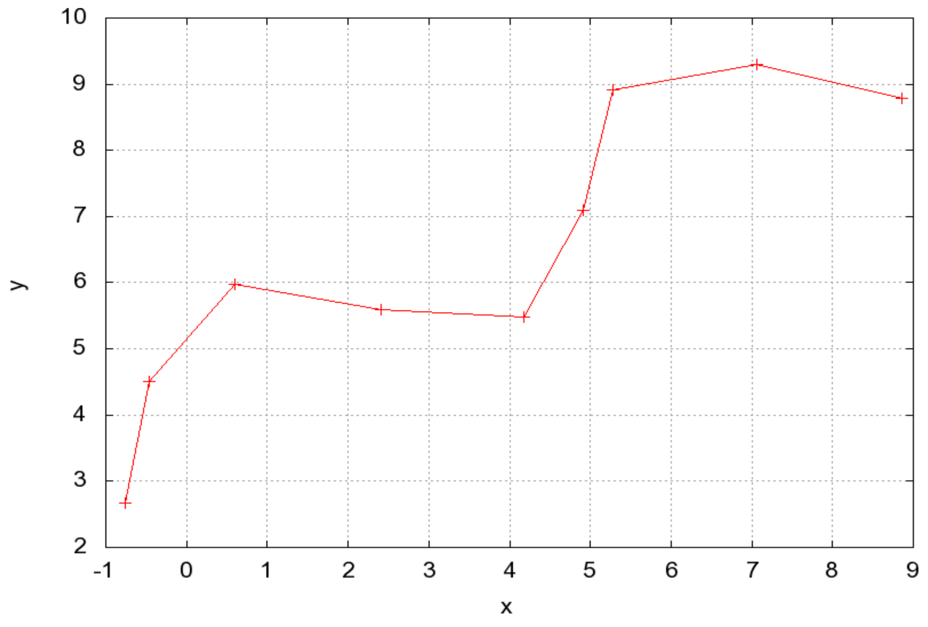

Figure 4.21: Reproduced points for the Spiral function. Value of the criterion function is 0.00186.

Triangle function

Results are presented in Figure 4.22.

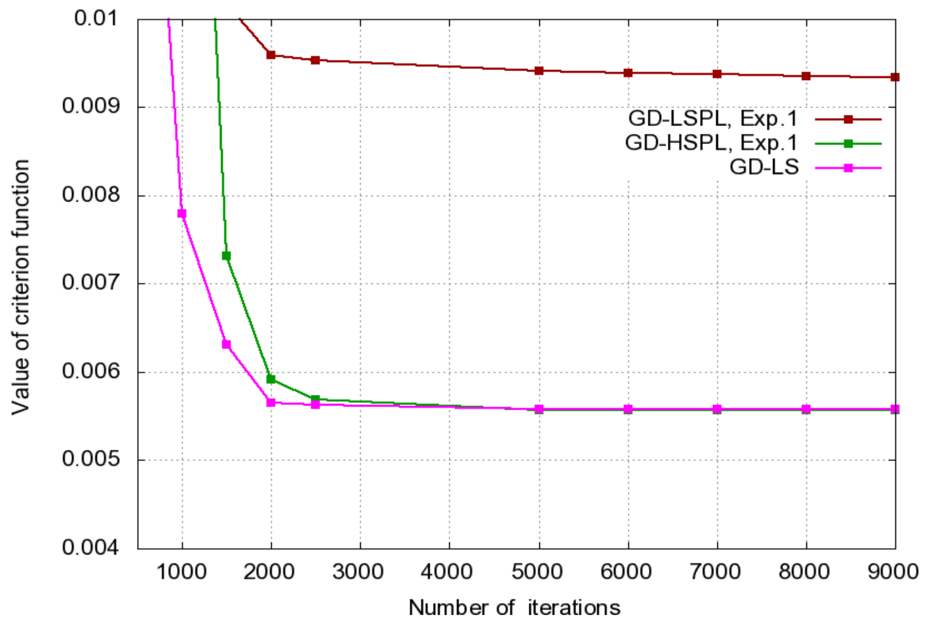

Figure 4.22: Experiment 1. Triangle function. Learning rate is applied to the criterion function globally. Results for GD-LS are plotted for comparison purposes.

In brief, again, the performance of GD-LS and GD-HSPL are very similar. However, GD-LS converges a bit faster towards the minimum. For comments on results achieved by GD-LSPL and GD-Classic see comments for the Spiral function.

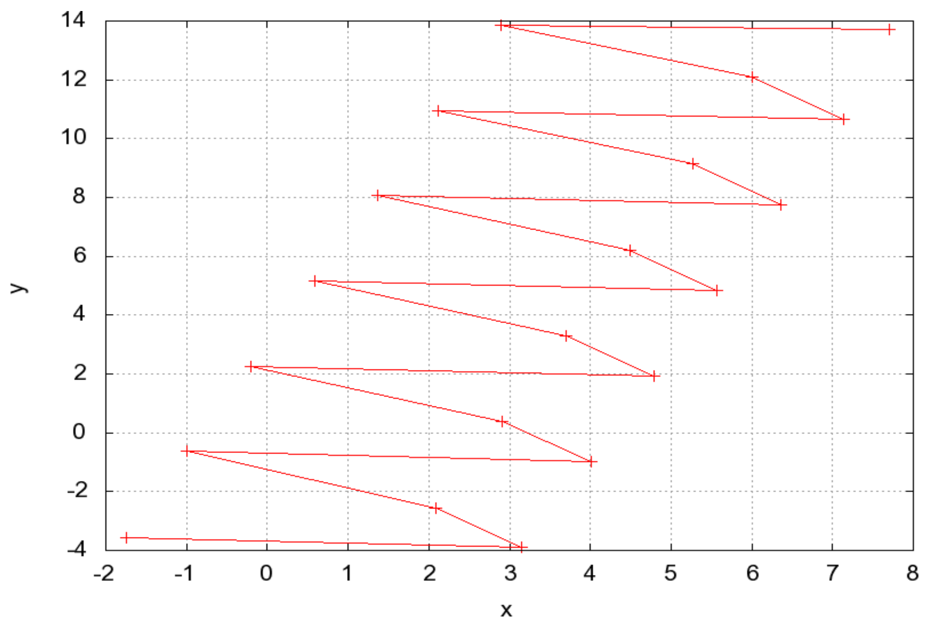

Figure 4.23: Reproduced points for the Triangle function. Value of the criterion function is 0.00175

The best value of the criterion function achieved during this experiment was 0.00175. Reproduced points for this result are presented in Figure 4.23.

Conclusion for Experiment 1

We have seen that when the learning rate is applied to the criterion function globally, the difference in performance of GD-HSPL and GD-LS is statistically insignificant. Convergence of GD-LSPL is slower and less accurate than convergence of GD-HSPL and GD-LS.

Also for this evaluation, we need to adjust manually the constant learning rate for GD-Classic to achieve fast and accurate convergence. GD-Classic with the learning rate close to the optimal constant learning rate converges as fast and accurate as GD-HSPL and GD-LS. However, with too large learning rate GD-Classic does not manage to convert, while with too small learning rate the convergence is very slow.

All results are to some degree random. A certain variation is also added by the choice of β for GD-LS. Nevertheless, the results are representative for measuring of the performance of the algorithms and give the bases for comparing them.

Reproduced points obtained by the experiment give an accurate representation of the distances between original points.

4.2.3 Experiment 2

In this experiment we compare the performance of GD-HSPL and GD-LSPL algorithms with the learning rate applied point-wise as described in 3.3(2).

Spiral function

Results are presented in Figure 4.24. Results for Experiment 1 and GD-LS are plotted for comparison purposes.

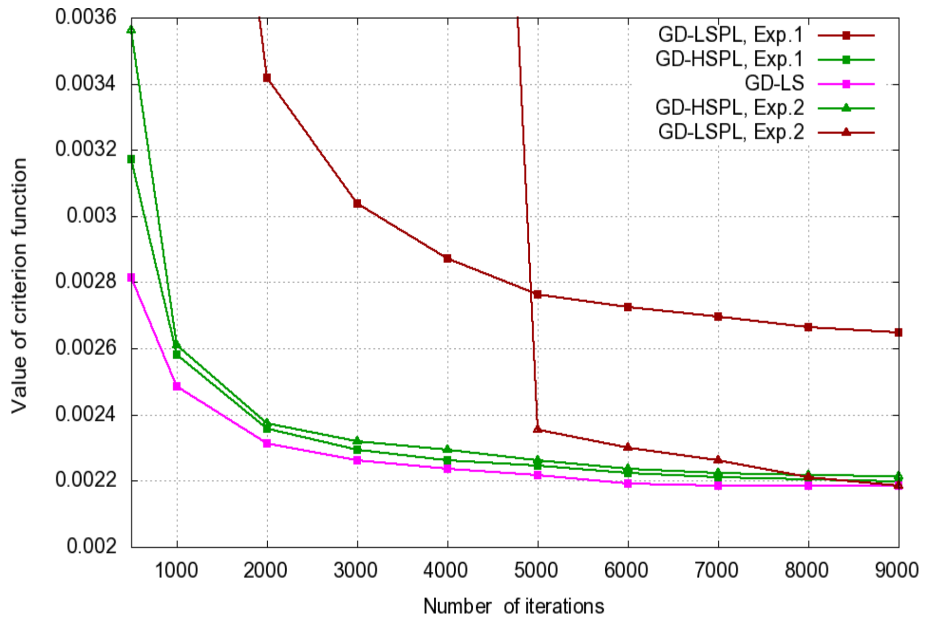

Figure 4.24: Spiral function. Experiment 2. Learning rate is applied point-wise. Results for Experiment 1 and GD-LS are plotted for comparison purposes.

As can be seen, GD-HSPL converges faster than GD-LSPL when the learning rate is applied point-wise. However, when we continue the experiment to 10000 iterations, the difference in convergence accuracy achieved by the algorithms is statistically insignificant.

We can also observe that GD-LSPL in Experiment 1 converges faster towards the minimum value of the criterion function than in Experiment 2. It is caused by the fact that for Experiment 2 a larger number of the learning rates (a separate learning rate for each point) need to be adjusted. For the Spiral function the learning rates need to be reduced from 500 to around 100. In detail, algorithms start with value of the learning rate defined by the maximal learning rate. For the maximal learning rate equal to 1000 the start value of the learning rates for GD-LSPL algorithm is 500. The optimal learning rate may vary during the convergence process and the algorithms may need several iterations to adjust the learning rates to the value giving the fast performance at the current position. In our case, the optimal learning rate in the start of the algorithms is around 100. Therefore, algorithms need to reduce the learning rate in the start of the execution process. The slowness in reduction of the learning rates to the optimal value is particularly noticeable for GD-LSPL changing the learning rate in small steps.

However, we can see that GD-LSPL in Experiment 2 converges more accurately to the minimum than in Experiment 1 when the algorithm approaches the minimum value of the criterion function. More accurate convergence of Experiment 2 is due to its capability to adapt the learning rate point-wise, meaning that the speed of the movement is suited separately for each point.

The difference in convergence accuracy for GD-HSPL in Experiment 1 and in Experiment 2 is statistically insignificant. However, Experiment 2 is a bit slower in convergence.

Triangle function

Results are presented in Figures 4.25. Results for Experiment 1 and GD-LS are plotted for comparison purposes.

For this function GD-HSPL converges faster and more accurately than GD-LSPL when the

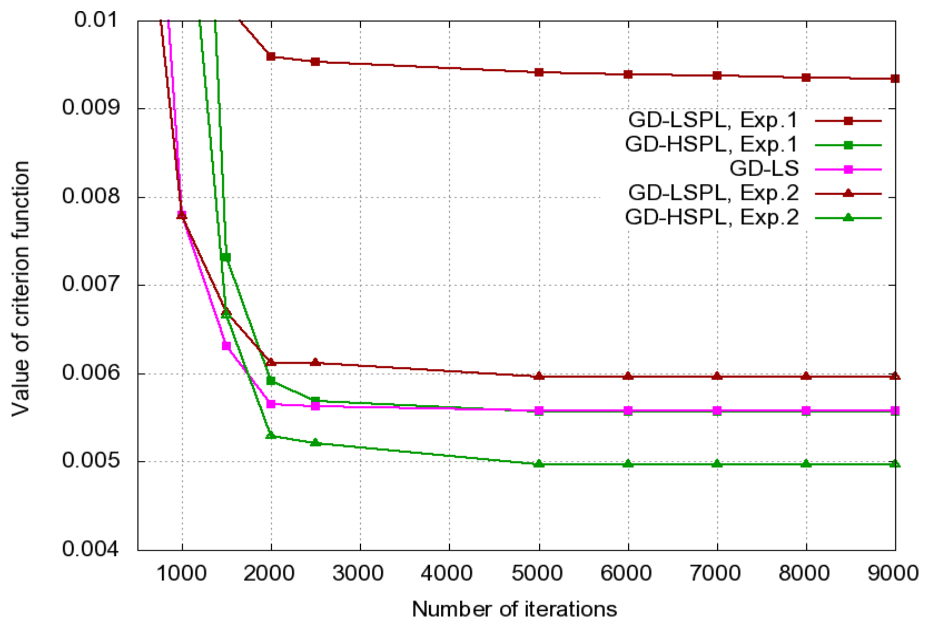

Figure 4.25: Triangle function. Experiment 2. Learning rate is applied point-wise. Results for Experiment 1 and GD-LS are plotted for comparison purposes.

learning rate is applied point-wise.

We can also observe that both GD-HSPL and GD-LSPL in Experiment 2 converges faster and more accurately to the minimum value of the criterion function than in Experiment 1. The faster convergence of Experiment 2 for the Triangle function compared to the Spiral function is caused by less difference in the maximal learning rate and the optimal learning rate at the start of the algorithms. For both functions we select the maximal learning rate to be 1000. However, the optimal start learning rate for the Spiral function is about 100, while for the Triangle function the optimal start learning rate is about 500. Since algorithms need to adjust the learning rate separately for each point, it takes longer time for the Spiral function to approach the minimum value of the criterion function.

It can also be seen that GD-HSPL in Experiment 2 converges more accurately than GD-LS for the Triangle function.

Conclusion for Experiment 2

We have observed that GD-HSPL converges faster than GD-LSPL when the learning rate is applied point-wise. The strategy applied in Experiment 2 gives more accurate convergence to the minimum value of the criterion function than the strategy applied in Experiment 1. If there is a big difference between the maximal learning rate and the optimal start learning rate, it may take long time for the algorithms in Experiment 2 to approach the minimum value of the criterion function. This is specially apparent for GD-LSPL changing the learning rate in small steps.

GD-HSPL with the learning rate applied point-wise instead for globally has shown to be able to converge more accurately than GD-LS.

4.2.4 Experiment 3

In this experiment we compare the performance of GD-HSPL and GD-LSPL algorithms with an unique learning rate applied to each pair of points (pair-wise) as described in 3.3(3). Thus, we randomly select two points at each iteration and calculate value of the criterion function only based on the distance between them. In this way each pair of points gets their own function to minimise. Anyway, at each iteration we check the value of the global criterion function, that is criterion function between all points, for comparison purposes.

Spiral function

First we used value of the criterion function between two points as measure for changing the learning rate. When the criterion function between two points was decreased, we increased the learning rate between them. Otherwise, we decreased it. The result achieved in this way is not competitive compared to the results achieved in Experiment 1 and Experiment 2. The minimal value of the criterion function achieved with the maximal learning rate equal to 1000 is equal to 0.02603. For Experiment 1 and Experiment 2 this value is equal to 0.00186.

Value of the criterion function between two points does not include the relation to the other points. Often the relation between two selected points is improved by the current move, while the relation to the other points is worsen. In this case value of the criterion function between two points is decreased while the value of the global criterion function is not. As a result, the learning rate between the selected points is increased whereas it should be decreased in order to have chance to lessen value of the global criterion function next time these two points are selected.

However, the positive feature of Experiment 3 is that it takes much less time to execute iterations, since we calculate value of the gradient based on criterion function only between two points and not between all points. We decided to change Experiment 3 to use the global criterion function (criterion function between all the points) as measure for changing the learning rate, while we still use the criterion function between the two selected points to calculate value of the gradient. By this setup we keep the advantage of the fast performance. For selected functions Experiment 3 with the new setup is at least 10 times faster than Experiment 1 or Experiment 2. At the same time we improve the criterion for changing the learning rate.

Results achieved by GD-HSPL and GD-LSPL algorithms for 9000 iterations are presented in Figures 4.26. Results for GD-LS, Experiment 1 and Experiment 2 are plotted for comparison purposes. Results for Experiment 3 are plotted for each 10. iteration since 10 iterations of Experiment 3 takes as long time as 1 iteration of Experiment 1 or Experiment 2.

As can be seen, GD-LSPL converges slightly more accurately than GD-HSPL when the learning rate is applied pair-wise. For Experiment 3 value of the gradient is calculated between two points. This measure is not as accurate as value of the gradient calculated between all the points. GD-HSPL changing the learning rate in larger steps is more influenced by the mentioned inaccuracy and converges less accurately than GD-LSPL changing the learning rate in smaller steps.

We can also observe that GD-LSPL in Experiment 3 with the new setup performs faster than in Experiment 1 and Experiment 2. However, the algorithm converges more accurately in Experiment 2 when it gets enough time.

GD-HSPL in Experiment 3 converges slower and less accurate than in Experiment 1 and in Experiment 2.

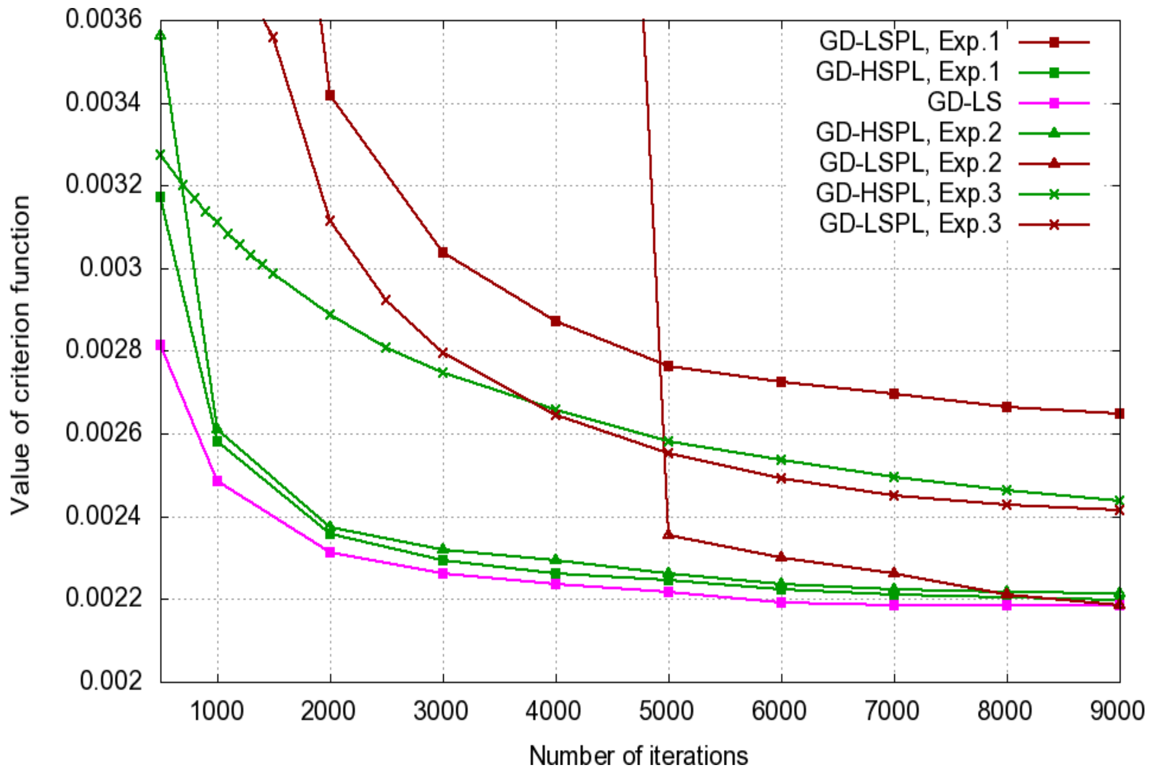

Figure 4.26: Experiment 3. Spiral function. Learning rate is applied pair-wise. Value of the global criterion function is used as measure for changing the learning rate. Results for Experiment 3 are plotted for each 10. iteration. Results for GD-LS, Experiment 1 and Experiment 2 are plotted for comparison purposes.

Triangle function

Again, we first used value of the criterion function between two points as measure for changing the learning rate. The minimal value of the criterion function achieved with the maximal learning rate equal to 1000 is equal to 0.00720. For Experiment 1 and Experiment 2 this value is equal to 0.00175 meaning a much more accurate convergence.

As for the Spiral function, we changed Experiment 3 to use global criterion function as measure for changing the learning rate, while we still use the criterion function between the two selected points to calculate value of the gradient.

Results achieved by GD-HSPL and GD-LSPL algorithms for 9000 iterations are presented in Figures 4.27. Results for GD-LS, Experiment 1 and Experiment 2 are plotted for comparison purposes. Results for the Triangle function are similar to results for the Spiral function. GD-LSPL converges slightly more accurate than GD-HSPL when the learning rate is applied pair-wise.

GD-LSPL in Experiment 3 with the new setup converges faster and more accurate than in Experiment 1. However, the convergence of GD-LSPL in Experiment 3 is worse in speed and accuracy than in Experiment 2. Again, GD-HSPL in Experiment 3 converges slower and less accurate than in Experiment 1 and Experiment 2.

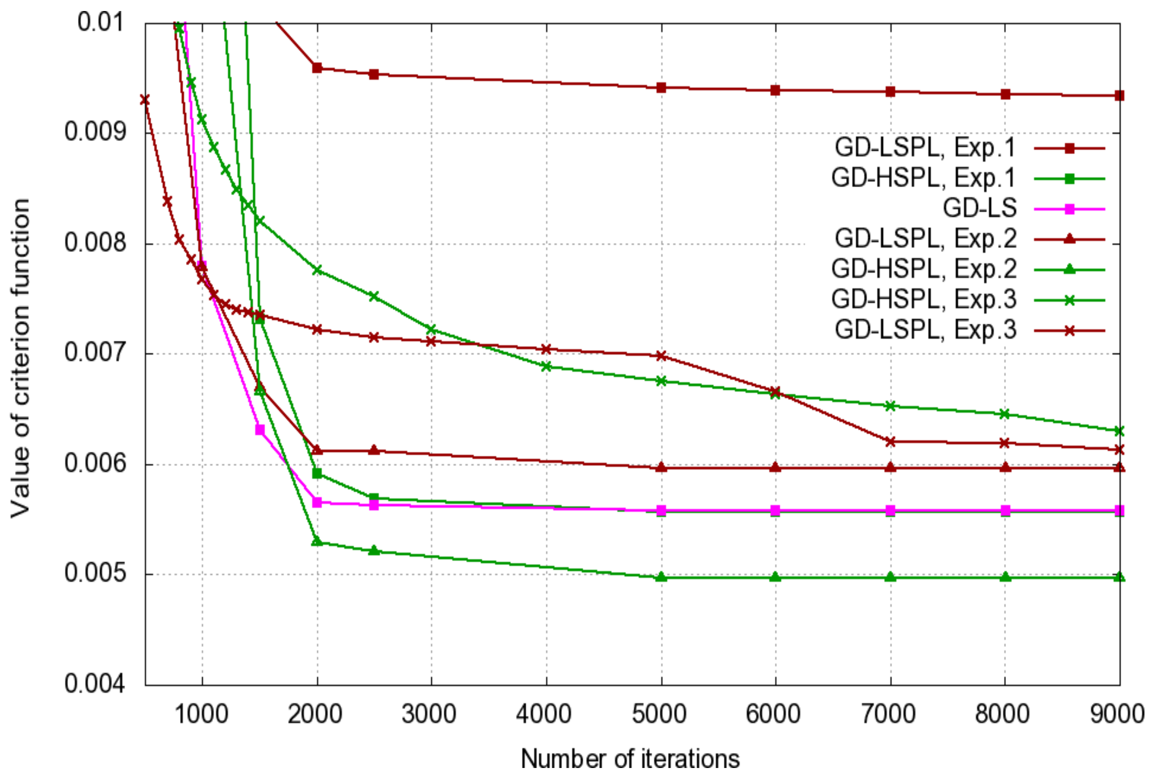

Figure 4.27: Experiment 3. Triangle function. Learning rate is applied to the pairs of points. Value of the global criterion function is used as measure for changing the learning rate. Results for Experiment 3 are plotted for each 10. iteration. Results for GD-LS, Experiment 1 and Experiment 2 are plotted for comparison purposes.

Conclusion for Experiment 3

The best result for Experiment 3 is achieved by using global criterion function as measure for changing the learning rate while we still use the criterion function between the two selected points to calculate value of the gradient.

GD-LSPL in Experiment 3 performs faster and more accurately than in Experiment 1 but less accurately than in Experiment 2. GD-HSPL in Experiment 3 converges slower and less accurate than in Experiment 1 and Experiment 2.

4.2.5 Conclusion for Artificial Multidimensional scaling problems

We have seen that when the learning rate is applied to function globally, the difference in performance of GD-HSPL and GD-LS is statistically insignificant. Convergence of GD-LSPL is slower and less accurate than convergence of GD-HSPL and GD-LS.

The strategy applying the learning rate point-wise for GD-HSPL and GD-LSPL makes convergence to the minimal value of the criterion function more accurate. Nevertheless, it may take longer time to adjust the learning rates and approach the minimum of the criterion function if the learning rate in the start of the algorithm is far from the optimal. The latter disadvantage is particularly noticeable for GD-LSPL changing the learning rate in small steps.

The strategy applying the learning rate pair-wise makes convergence of GD-LSPL faster and more accurate compared to the strategy applying the learning rate globally to the function. However, this method does not improve the convergence accuracy of GD-LSPL with the learning rate applied point-wise. The strategy applying the learning rate pair-wise for GD-HSPL neither improve speed nor accuracy of the strategies applying the learning rate globally and point-wise.

To conclude, we submit that the strategy applying the learning rate point-wise achieves the most accurate performance for GD-HSPL and GD-LSPL in solving the artificial MDS problems.

GD-HSPL and GD-LSPL (especially GD-HSPL) may improve GD-based minimisation of the criterion function of the artificial MDS problem. Compared to GD-Classic there is a clear improvement, since GD-LSPL and GD-HSPL adjust the learning rate during execution. Compared to GD-LS, GD-LSPL performs slower. The performance of GD-HSPL applying the learning rate point-wise is close to the performance of GD-LS and has shown the ability to improve the performance of GD-LS several times. In addition, using GD-HSPL and GD-LSPL reduces randomness introduced by the choice of β for GD-LS algorithm.

However, the improvement introduced by GD-HSPL and GD-LSPL in solving the artificial MDS problems is often statistically insignificant. The latter improvement is not large compared to the improvement achieved in minimising mathematical functions. The difference is that in minimising mathematical functions randomness is introduced to the problem data, while in solving the artificial MDS problems randomness is located in the algorithm itself. It looks like GD-LS handle randomness in the algorithm quite well. Therefore, it is difficult to improve its performance by the learning algorithms.

4.3 Classifying Word-Of-Mouth Discussions

This section presents experiments evaluating the proposed algorithms applied to GD based MDS method for Classifying Word-Of-Mouth Discussions.

4.3.1 Experiment setup

In order to create input to the MDS analysis documents are presented by their Vector space model as described in 2.3.2. Each document is represented by a vector of terms. We use Frequency-Inverse Document Frequency (TF-IDF) method to calculate term weights. The Cosine Similarity is used for comparing documents. Dissimilarity matrix needed as input to the MDS analysis is constructed using the inverse of the cosine similarity between document vectors. We calculate dissimilarity as follows.

$$dissimilarity = \frac{1}{similarity} \tag{4.3}$$

Our attempt is to present documents in 2-dimensional mapping reflecting similarities between the documents.

WoM discussions concerning the following topics and subtopics have been used in the experiments.

1. Health
   - Pregnancy - 5 documents
   - Cancer - 5 documents
   - Weight reduction - 5 documents
2. Environment
   - Alternative energy - 5 documents
   - CO_2 - 4 documents
3. Religion
   - Christian faith - 4 documents
   - Islam - 9 documents
4. Travel
   - Prague - 4 documents
   - London - 5 documents
   - Space travel - 6 documents

The setup for the algorithms and experiments are the same as in the sub-chapter 4.2. The criterion function for MDS analysis S is calculated according to 2.10. The maximal learning rate is selected to be 100.

We generate a set of initial positions for each document drown from the interval [0;200]. The ensemble average of 100 executions is found.

As explained in 4.2, the tree ways of applying the learning rate which we evaluate in this section are not applicable for GD-LS and GD-Classic algorithms. However, we include the results for these algorithms for comparison purposes.

4.3.2 Experiment 1

In this experiment we compare the performance of GD-LSPL and GD-HSPL algorithms with the learning rate applied globally to the criterion function as described in 3.3(1).

Results are presented in Figure 4.28.

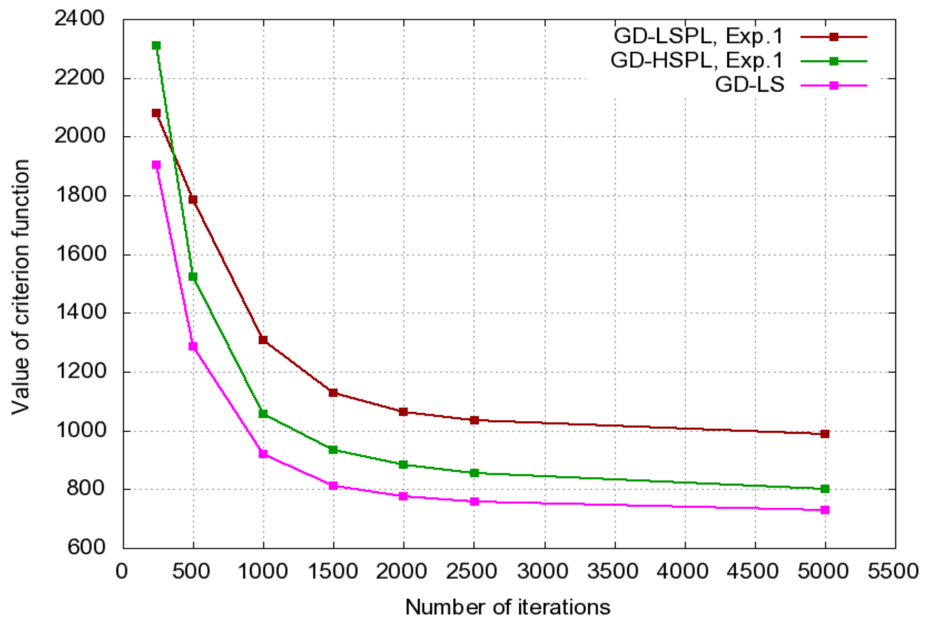

Figure 4.28: Experiment 1. Classifying WoM discussions. Learning rate is applied globally to the criterion function. Results for GD-LS are plotted for comparison purposes.

As can be seen, GD-HSPL converges faster and more accurately than GD-LSPL when the learning rate is applied globally to the criterion function. However, the performance of GD-LS is even faster and more accurate than the performance of GD-HSPL.

Result for GD-Classic is not included because the algorithm does not manage to converge with the learning rate equal to 100. Performance of GD-Classic with the learning rate close to the optimal constant learning rate equal to 10 is still slower and less accurate than the performance of GD-HSPL and GD-LS. This shows that using the optimal constant learning rate strategy does not always give as fast and accurate result as strategies adjusting the learning rate during execution.

One of the smallest values of the criterion function achieved by this experiment is $S = 691.96$. Reproduced positions for documents corresponding to this value of the criterion function are presented in Figure 4.29. As can be seen, documents belonging to the same topics with the similar content are mostly placed close to each other. In general, we can say that performed GD based MDS analysis reflects very well distances $\{\delta_{ij}\}$ between documents calculated by the Cosine Similarity.

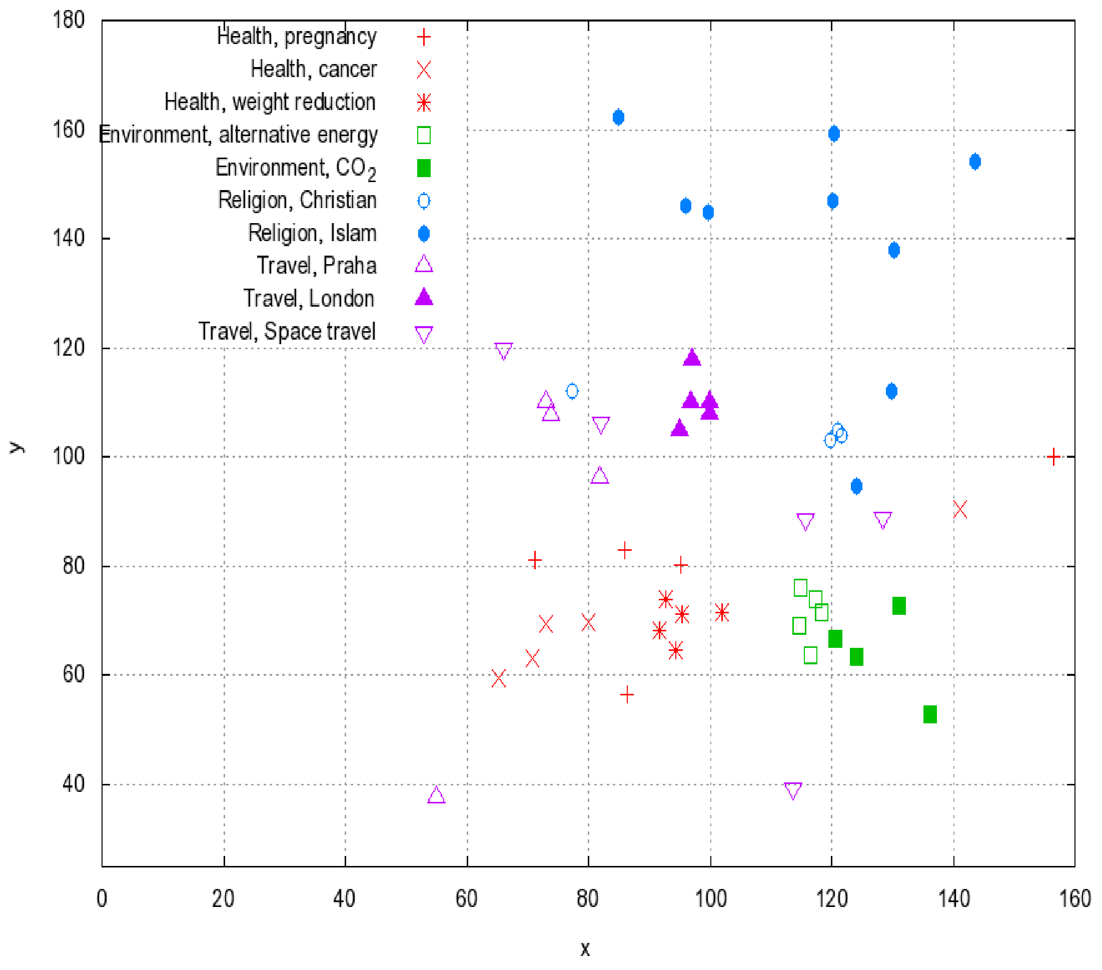

Figure 4.29: 2-dimensional mapping of the WoM discussions with topics Health, Environment, Religion and Travel. Value of the criterion function is 691.96

4.3.3 Experiment 2

In this experiment we compare the performance of GD-HSPL and GD-LSPL algorithms with the learning rate applied point-wise as described in 3.3(2). For Classifying WoM discussions points are reproduced positions of the documents in 2-dimensional mapping. Results are presented in Figure 4.30. Results for Experiment 1 and GD-LS are plotted for comparison purposes.

As can be seen, GD-HSPL converges faster and more accurately than GD-LSPL when the learning rate is applied point-wise. The performance of GD-LSPL is slow due to the slow adjusting of the learning rates described in 4.2.3. GD-LSPL in Experiment 2 also performs much slower than in Experiment 1.

GD-HSPL in Experiment 2 converges more accurately towards the minimum value of the criterion function than in Experiment 1, even the convergence of the algorithm in Experiment 2 is slower in the beginning. The difference in accuracy achieved by GD-HSPL in Experiment 2 and GD-LS is statistically insignificant. However, GD-LS converges faster towards the minimum of the criterion function.

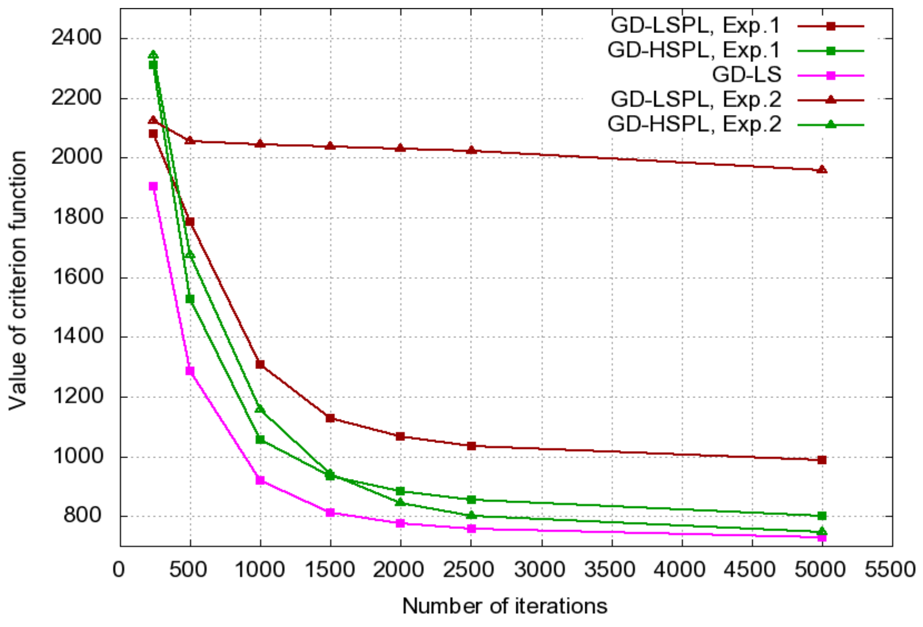

Figure 4.30: Experiment 2. Classifying WoM discussions. Learning rate is applied point-wise. Results for Experiment 1 and GD-LS are plotted for comparison purposes.

4.3.4 Experiment 3

In this experiment we compare the performance of GD-HSPL and GD-LSPL algorithms with an unique learning rate applied to each pair of documents (pair-wise) as described in 3.3(3). Thus, we randomly select two documents at each iteration and calculate value of the criterion function only based on the dissimilarity measure between them.

Based on the experience from solving the artificial MDS problems, we use global criterion function as measure for changing the learning rate while we use the criterion function between the two selected documents to calculate value of the gradient.

Results are presented in Figure 4.31. Results for GD-LS, Experiment 1 and Experiment 2 are plotted for comparison purposes. Result for GD-LSPL in Experiment 2 is not plotted because it is of scale compared to the other results (value of the criterion function is around 2000). Results for Experiment 3 are plotted for each 10. iteration since 10 iterations of Experiment 3 takes as long time as 1 iteration of Experiment 1 or Experiment 2.

As can be seen, GD-LSPL performs faster and more accurate than GD-HSPL when the learning rate is applied pair-wise. The reason for better performance of GD-LSPL is inaccuracy in calculation of the gradient discussed in 4.2.4.

We can also observe that GD-LSPL in Experiment 3 converges much faster and accurately than in Experiment 1 and in Experiment 2. GD-LSPL in Experiment 3 also converges faster than GD-LS. The difference in accuracy achieved by GD-LSPL in Experiment 3 and GD-LS is statistically insignificant.

GD-HSPL in Experiment 3 performs faster than in Experiment 1 and Experiment 2. The difference in accuracy achieved by GD-HSPL in Experiment 2 and Experiment 3 is statistically insignificant.

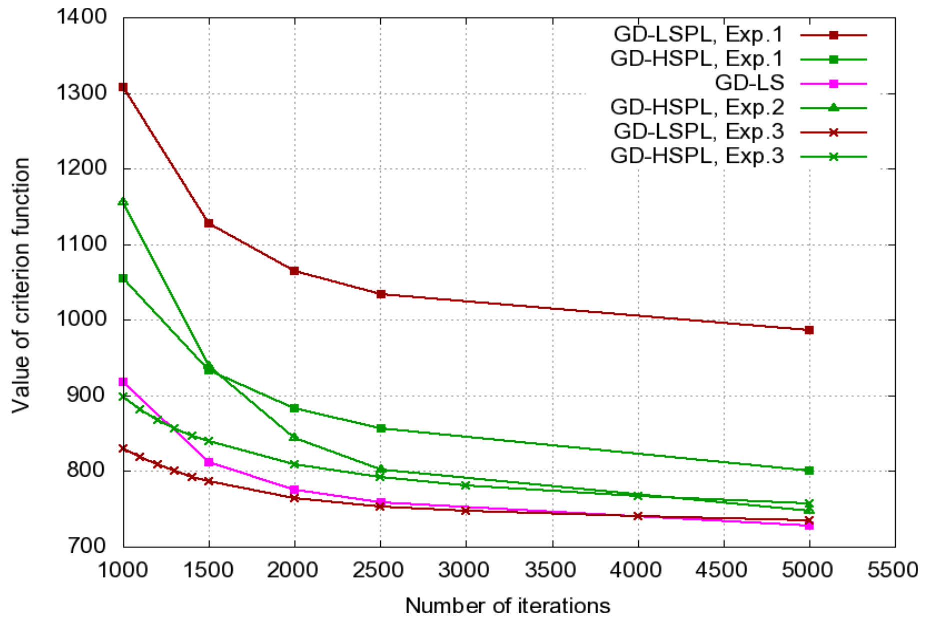

Figure 4.31: Experiment 3. Classifying WoM discussions. Learning rate is applied pair-wise. Results for Experiment 3 are plotted for each 10. iteration. Results for GD-LS, Experiment 1 and Experiment 2 are plotted for comparison purposes.

4.3.5 Conclusion for Classifying Word-Of-Mouth Discussions

Applying the learning rate point-wise strategy for both GD-LSPL and GD-HSPL performs slower in Classifying WoM discussions than in solving artificial MDS problems. As has been mentioned in 4.2.3, this strategy may need long time to adjust the learning rates, particularly for GD-LSPL. Since the amount of points has been increased in Classifying WoM discussions, the number of the learning rates has increased equally. Therefore, it is not surprising that the strategy applying the learning rate point-wise performs slower. In practise, when number of points is typically much larger than has been used in our evaluation, GD-LSPL applying the learning rate point-wise will not be a competitive solution.

Another difference compared to solving artificial MDS problems is very good performance of the strategy applying the learning rate pair-wise. Both for GD-HSPL and GD-LSPL applying the learning rate pair-wise is the fastest performing strategy. However, this strategy results in faster and more accurate convergence for GD-LSPL compared to GD-HSPL. This is due to inaccuracy in calculation of the gradient that influences GD-HSPL more than GD-LSPL since GD-HSPL changes the learning rate in larger steps. In addition, the strategy applying the learning rate pair-wise results in faster convergence for GD-LSPL compared to GD-LS. The difference in convergence accuracy between these algorithms is statistically insignificant.

The reason for the fast and accurate performance of the strategy applying the learning rate pair-wise is probably its high execution speed. Algorithms using the latter strategy are able to execute at least 10 times as many iterations as algorithms using other strategies within the same amount of time. Therefore, the strategy applying the learning rate pair-wise offers a larger amount of iterations for adjustment of the learning rates, even though there is a separate learning rate for each pair of documents.

The reason for improved performance of the strategy applying the learning rate pair-wise compared to evaluation of the artificial MDS problems is increased number of points in MDS analysis.

When number of points increases, the strategy applying the learning rate point-wise becomes slower, while the strategy applying the learning rate pair-wise becomes the fastest one.

To conclude, we submit that GD-HSPL and GD-LSPL may improve GD based MDS method in Classifying Word-Of-Mouth Discussions. Compared to GD-Classic there is a clear improvement since GD-LSPL and GD-HSPL adjust the learning rate during execution. As a result, algorithms converge faster and more accurately than GD-Classic with the optimal constant learning rate.

GD-LSPL with the learning rate applied pair-wise performs faster compared to GD-LS. The difference in convergence accuracy between them is statistically insignificant. Performance of GD-HSPL with the learning rate applied pair-wise is also close to the performance of GD-LS. Therefore, both GD-HSPL and GD-LSPL has potential in improving the performance of GD-LS in Classifying Word-Of-Mouth Discussions.

Chapter 5

Conclusion and future work

In this chapter we present the major conclusions for the thesis and possible directions for the future work.

In this thesis we have enhanced the classical Gradient descent (GD) algorithm by incorporating Linear SPL and Hierarchical SPL schemes for iterative determining the learning rate. As a result, two new GD based algorithms, GD-LSPL and GD-HSPL, have been developed. Our hypothesis is that GD-LSPL and GD-HSPL improve the performance of GD based optimisation in stochastic environments.

For evaluation purposes GD-LSPL and GD-HSPL algorithms have been compared with each other, with GD with constant learning rate (GD-Classic) and with GD with Line search (GD-LS) algorithms. The algorithms have been empirically evaluated in three evaluation environments. They are minimising mathematical functions, artificial Multidimensional Scaling (MDS) problems as well as a practical MDS problem concerning the Classification of Word-Of-Mouth (WoM) discussions.

We have proposed three different schemes for applying GD-LSPL and GD-HSPL algorithms when identifying the learning rate in the GD based MDS. In brief, the schemes consist of either identifying a global learning rate that is applied to all the points, an unique learning rate for each individual point (point-wise), or an unique learning rate for every pair of points (pair-wise). These proposed schemes have been rigorously evaluated for artificial MDS problems and for classifying WoM discussions.

We have also proposed different ways of selecting points and evaluating the criterion function together with different criteria for updating the learning rate. The usual way is to select one point at each iteration of the GD based MDS algorithm and evaluate the criterion function together with the gradient based on all the points. When processing a single point at a time, it is natural to modify the position of the processed point based on the value of the criterion function that is sought minimised. Value of the same criterion function is also used to decide whether the learning rate should be increased or decreased. The other option is to select two points at each iteration and evaluate the criterion function and gradient only based on the relative position of the two points. Value of the criterion function between two selected points is also used to decide whether the learning rate should be increased or decreased. It turns out that processing pairs of points at a time option is more computationally efficient compared to the first option. However, the learning accuracy is reduced. Thus, we have identified a third option that combines the computational efficiency of the second option with the accuracy of the first option. We quite simply select two points and use the criterion function isolated to these two points for calculation of the gradient, while we use the criterion function for all the points for changing the learning rate.

5.1 Summary of results

The reported empirical results have shown that GD-LSPL and GD-HSPL algorithms are efficient and adaptive for operation in stochastic environments. GD-LSPL is generally slower than GD-HSPL. This is due to the difference in the SPL schemes. Since Hierarchical SPL scheme used in GD-HSPL changes the learning rate in larger steps, this algorithm generally adapts faster in stochastic environments.

Compared to GD-Classic, GD-LSPL and GD-HSPL algorithms introduce a clear improvement since they regulate the learning rate during execution. As a result, GD-HSPL and GD-LSPL algorithms has shown ability to converge faster and more accurately than GD-Classic with the optimal constant learning rate giving the fastest convergence for GD-Classic. In addition, the new algorithms eliminate the need for manual adjustment of the learning rate in GD-Classic.

Minimising mathematical functions

In minimising mathematical functions, when we increase randomness or accuracy during the experiments, GD-HSPL shows the fastest and the most accurate performance. Exception from this rule is the case when randomness or accuracy is very high and, in addition, local minima of the function being minimised are located very close to each other. For this kind of functions GD-LSPL is obviously a better alternative. Moreover, GD-HSPL is the least influenced by variation in the start value and the most frequently converging to the global minimum of the function algorithm.

In minimising mathematical functions GD-LS has shown poor adaption in stochastic environments. This algorithm converges slowly compared to the other algorithms when randomness or accuracy increases during evaluation. GD-LS reduces the learning rate each time a new calculated position does not introduce reduction to the value of the function being minimised. When randomness increases, the number of such unsuccessful attempts rises. Therefore, the learning rate of GD-LS often becomes unnecessarily low giving very slow convergence.

Thus, we submit that GD-LSPL and GD-HSPL improve the performance of GD based optimisation in minimising mathematical functions.

MDS problems

Selecting one point at each iteration and applying learning rate point-wise has shown to be the most accurately converging strategy for both GD-HSPL and GD-LSPL in solving artificial MDS problems. However, this way of applying the learning rate may be very slow if the learning rate in the start of the algorithm is far from the optimal learning rate giving the fastest convergence. In this case it may take longer time to approach the minimum of the criterion function since there is a need to adjust the learning rates individually for each point. The latter disadvantage is particularly noticeable for GD-LSPL changing the learning rate in small steps.

The slow performance of the applying learning rate point-wise strategy has been confirmed by findings from the evaluation of the WoM discussion classification. In this case the number of points has increased resulting in increased number of the learning rates that are need to be adjusted. The increased number of the learning rates has led to slowing down the convergence process. In practise, when number of points is typically much larger than has been used in our evaluation, GD-LSPL applying the learning rate point-wise will not be a competitive solution.

The quickest approach for both GD-HSPL and GD-LSPL in Classifying WoM discussions is applying the learning rate pair-wise. The fastest performance is achieved by using the criterion function between the two selected points for calculation of the gradient, while using the criterion function for all the points for changing the learning rate. We believe that the reason for this fast performance is the high execution speed of the scheme allowing at least 10 iterations during each iteration performed by the other schemes. The fast execution of iterations provides more time to adjust the learning rates, even though there is a larger amount of the learning rates compared to the other schemes. In the latter scheme value of the gradient is calculated between two points. This measure is not as accurate as the value of the gradient calculated between all points. GD-HSPL changing the learning rate in larger steps is more influenced by the mentioned inaccuracy. Therefore, this scheme performs faster and more accurate for GD-LSPL than for GD-HSPL. The advantage of the strategy applying the learning rate pair-wise increases when the amount of points in MDS analysis increases.

5.2 Conclusion

The performance of GD-HSPL and GD-LSPL in solving MDS problems is close to the performance of GD-LS and has shown the ability in improving the performance of GD-LS, specially when number of points in MDS increases. In addition, using GD-HSPL and GD-LSPL eliminates randomness introduced by the choice of β for GD-LS algorithm. However, both in solving the artificial MDS problems and in Classifying WoM discussions GD-LS performs very well and the improvement achieved by GD-HSPL and GD-LSPL compared to GD-LS is often statistically insignificant. The reason for good performance of GD-LS in solving MDS problems is that randomness of MDS is found in the algorithm itself compared to minimising mathematical functions where randomness is introduced to the problem data.

In addition to show potential in improving performance of GD-LS in solving MDS problems, GD-HSPL and GD-LSPL obviously have shown a great improvement in minimising mathematical functions. For this reason we submit that the above findings are consistent with the main hypothesis being true. GD-LSPL and GD-HSPL improve the performance of GD based optimisation in stochastic environments.

5.3 Further work

Although we have already evaluated performance of GD-HSPL and GD-LSPL both in solving artificial MDS problems and in using MDS method for Classifying WoM discussions, there is still room for further investigations. For example, MDS problems used for evaluation of the algorithms include between 9 and 52 points. Real life problems typically include a larger amount of points. Some of the unanswered questions are listed below.

- How will the larger number of points in MDS influence the performance of the GD-HSPL and GD-LSPL compared to GD-LS?
- How will the larger number of dimensions (length of documents in Classifying WoM discussions) influence the performance of the algorithms?

- Will increasing the number of points make applying the learning rate point-wise in MDS unusable also for GD-HSPL?
- How will the larger number of points in MDS influence the performance of GD-HSPL compared to GD-LSPL when we use the strategy applying the learning rate pair-wise? Is it possible to exploit the advantage of the generally faster performing GD-HSPL?
- How much faster is applying the learning rate pair-wise strategy compared to the other strategies?
- Will GD-LSPL applying the learning rate pair-wise work faster and more accurate than GD-LS when the number of points increases?

We also believe that GD-HSPL and GD-LSPL algorithms can be utilised in other GD based applications, particularly when randomness of the task is located in the problem data. For this purpose the further investigations including several GD based applications can be of great advantage.

Bibliography

[1] Magnus Bjerkeseth and Nadja Vidnes. Hierarchical stochastic point location. Technical report, University of Agder, Faculty of Engineering and Science, 2009. Project report for IKT508 in Autumn 2009.

[2] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. New York: John Wiley S Sons, Inc, 2000.

[3] Au. N I M Gould and Au. S Leyffer. *An introduction to algorithms for nonlinear optimization*, pages 109–197. Springer Verlag, 2003.

[4] Granino Arthur Korn, Theresa M.; Korn. *Mathematical Handbook for Scientists and Engineers: Definitions, Theorems, and Formulas for Reference and Review*, page 157160. New York: Dover Publications, 2000.

[5] J. B. Kruskal. Nonmetric multidimensional scaling. *Psychometrika*, 29:1–27, 1964.

[6] Zuoyuan Liang and B. John Oommen. Classifying literature using multidimensional scaling.

[7] B. John Oommen. Stochastic searching on the line and its applications to parameter learning in nonlinear optimization. *IEEE Transactions on Systems, Man, and Cyberneticspart B: Cybernetics*, 27:733–739, 1997.

[8] B. John Oommen, Ole-Christoffer Granmo, and Zuoyuan Liang. A novel multidimensional scaling technique for mapping word-of-mouth discussions. In *Opportunities and Challenges for Next-Generation Applied Intelligence*, pages 317–322. Springer Berlin / Heidelberg, 2009.

[9] Nicolas Le Roux. Using gradient descent for optimization and learning. Technical report, University of Cambridge, May 2009. Available on Talks.cam.

[10] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. New York: Mc-Graw Hill Book Company, 1983.

[11] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. of the ACM*, 18(11):613–620, 1975.

[12] G. Salton, C. S. Yang, and C. Yu. A theory of term importance in automatic text analysis. Technical report, Ithaca, NY, USA, 1974.

[13] Zhen-Jun Shi and Jie Shen. Step-size estimation for unconstrained optimization methods. *Comput. Appl. Math.*, 24(3):384–390, Sept./Dec. 2005.

[14] Gilbert Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 2009.

[15] W. S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17:401–419, 1952.