Comparison of Gradient Descent and Block Coordinate Gradient Descent methods through Semi-supervised learning problem

Dejan Dichoski,
Suleyman Erim,
Marija Cveevska
Department of Mathematics,
Università degli Studi di Padova
email: dejan.dichoski@studenti.unipd.it
sueleyman.erim@studenti.unipd.it
marija.cveevska@studenti.unipd.it

Professor
Francesco Rinaldi
Optimisation for Data Science
Department of Mathematics,
Università degli Studi di Padova,
email: rinaldi@math.unipd.it

Acquiring data has become relatively affordable nowadays, but a major challenge lies in obtaining the necessary labels for a learning problem. In many instances, only a limited amount of labeled data can be obtained while there is a large volume of unlabeled data available. Semi-supervised classification comes into play in such situations as it falls between supervised learning, where all data labels are known, and unsupervised learning, where no labels are present.

Keywords: GD, BCGD, semi-supervised, optimisation

1 Introduction

In this study, various optimization algorithms including standard gradient descent, block coordinate gradient descent (BCGD) with randomized rule and Gauss Southwel rule, are used to address a semi-supervised learning problem. To accomplish this, a set of random points and labels are initially generated, with most of the labels being later discarded. The accuracy of data point classification and the loss function are computed for each algorithm. Additionally, an analysis of the loss function, number of iterations and CPU time is conducted. Finally, the effectiveness of the algorithms is evaluated using a real-world dataset.

1.1 Problem Definition. We are faced with a binary classification challenge, where the number of points in each cluster is approximately equal. Out of all the examples, l are labeled: $(\bar{x}^i, \bar{y}^i), i = 1, ..., l$, and u are unlabeled: $x^j, j = 1, ..., u$. The objective is to determine the labels y^j for the unlabeled points, with the assumption that similar features will correspond to similar labels. To achieve this we rely on the concept of "similarity distance," which suggests that if the features are close to each other in space, there is a higher probability that they belong to the same class. The Euclidean distance is utilized as a measure of similarity between two points, with w_{ij} representing the similarity between labeled examples i and unlabeled examples j, and $\bar{w}_{ij}$ denoting the similarity between unlabeled examples. The primary task is to minimize Eq.(1):

$$\min_{y\in\mathbb{R}^u} \sum_{i=1}^{l}\sum_{j=1}^{u} w_{i,j}(y^j - \bar{y}^i)^2 + \frac{1}{2}\sum_{i=1}^{u}\sum_{j=1}^{u} \bar{w}_{i,j}(y^i - y^j)^2$$

The gradient of Eq.(1) with respect to y^j is represented by Eq.(2):

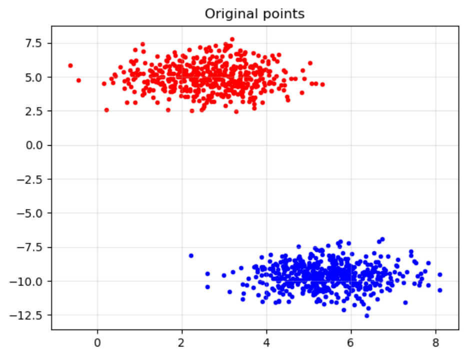
$$\nabla y^j f(y) = 2\sum_{i=1}^{l} w_{ij}(y^j - \bar{y}^i) + 2\sum_{i=1}^{u} \bar{w}_{ij}(y^j - y^i)$$

In order to classify the unlabeled examples, we employ gradient descent and block gradient descent techniques that yield minimum to the loss function and utilizing Eq.(2) to compute the gradient.

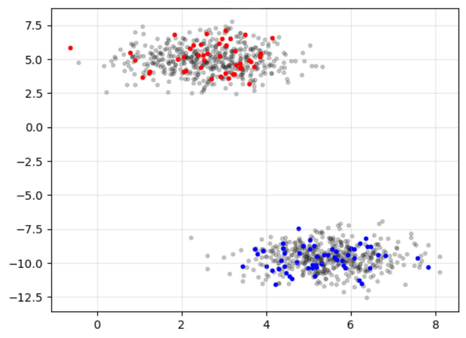
2 Generate Dataset

In this section we are going to randomly generate a set of points in 2D and give labels to a small subset of those points.

2.1 Generation of Data. A set of 1000 points were generated randomly and labeled as either 1 or -1 to ensure equal distribution between two clusters. The relative distance between the clusters was chosen based on the diagram in Fig. 1. 10% of the points were randomly selected to preserve their original labels, while the rest had their labels removed and randomly initialized labels to 1 and -1. The data provided to the algorithms includes a dictionary of labeled points and a dictionary of unlabeled points.



(*a*)



(*b*)

Fig. 1 Generated clusters of points (a) and the same clusters with labeled and unlabeled points (b)

3 Algorithms

We start with analyzing general scheme of a gradient-based minimization method:

```
Alg. 1: GRADIENT METHOD
Choose an initial approximation point $x \in \mathbb{R}^u$
for $k = 1, \ldots, k_{\max}$
    Check stopping Conditions
    Compute Descent Direction $d_k$
    Compute Stepsize $\alpha_k$
    Update Rule: $x_{k+1} = x_k - \alpha_k d_k$
End for
```

These algorithms can be modified by changing some parameters which affect their convergence.

Stopping Condition

The stopping condition in optimization algorithms is a criterion that determines when to stop iterating and return the current solution.For our Algorithms we use different early stopping conditions.

- The norm of the Gradient to be lower than a threshold,
- Reaching the specified number of iterations,
- The loss function increases more than 50 iterations consecutively,
- The Loss reaches Inf or NAN,
- Reaching the plateau of the loss curve, i.e., $\delta_i/\delta_{i-1} <$ 5% where $\delta_i = \text{Loss}_i - \text{Loss}_{i-1}$

Step size also known as learning rate, is a hyperparameter used in optimization algorithms to determine the size of the step taken in each iteration towards the minimum of the objective function. We use different learning rates for the Algorithms and they are the following:

- Fixed Step Size - constant value used for the step size parameter (usually in range 1e-3 to 1e-5)
- 1/L - The Lipschitz constant L of a function f with a Hessian matrix H is defined as:
  $L = \max_{x \in \mathbb{R}^n} |H(x)|$
  where $L$is the largest eigenvalue of the matrix.
- Armijo Rule - The Armijo rule involves starting with an initial step size and then iteratively reducing it until a suitable step size is found. At each iteration, the rule checks if the new point obtained by taking a step with the current step size satisfies a sufficient decrease condition. This condition ensures that the new point is "close enough" to the minimum point, and if not, the step size is reduced.

  $$\text{While } f(x_k + \alpha d_k) > f(x_k) + \gamma \alpha \nabla F(x_k)^T d_k$$

  $$\alpha = \delta^m \Delta_k$$

  The values for the hyperparameters that we use are: $\alpha = 0.05, \delta = 0.95, \gamma = 0.49$
- $1/L_i$, where Li is the Lipschitz constant related to each block. In our case, Li is a vector of size u (number of unlabeled samples), corresponding to the diagonal of the Hessian matrix.

```
Alg. 2: Block Coordinate Gradient Descent
Choose a point $x_1 \in \mathbb{R}^n$
for $k = 1, \ldots$
    If $x_k$ satisfies some specific condition,
    then STOP
    Set $y_0 = x_k$, pick blocks $S \subseteq 1, \ldots, b$, and
    set $l = |S|$
    For $i = 1, \ldots, l$
    Select $j_i \in S$ and set
    $y_i = y_{i-1} - \alpha_i U_{j_i} \nabla_{j_i} f(y_{i-1})$
    with $\alpha_i > 0$ calculated using a
    suitable line search
    Set $x_{k+1} = y_l$
End For
```

3.1 Gradient Descent. The conventional gradient descent algorithm computes the gradient for a point and modifies the label of that point before computing the gradient for the next point. This process is repeated until a stopping condition is met.

3.2 Block Coordinate Gradient Descent (BCGD). BCGD is an extension of the gradient descent algorithm, which iteratively updates the parameters of a model to find the optimal values that minimize the objective function. In BCGD, instead of updating all parameters at once, the algorithm updates a block of parameters (coordinates) at each iteration. The size of the block can be predetermined, and it is usually smaller than the total number of parameters. By updating only a small block of parameters at each iteration, the algorithm reduces the computational cost and memory requirements, making it efficient for large-scale optimization problems. In our case, we kept a fixed block size of 1, meaning that only one example had its label updated per iteration.
BCGD with randomized selection - is a variation of the BCGD algorithm and it updates the label of each randomly selected point (with uniform probability) after computing its gradient, and proceeds with the next point until the stopping condition is satisfied. Since the number of unlabeled points is u, the probability of selecting a certain point is 1/u.
Gauss - Southwell BCGD - the block of parameters is selected based on a Gauss-Southwell rule:

$$i_k = \arg \max_{j \in \{1,\dots,b\}} |\nabla_j f(x_k)|$$

Also we considered a variant, which showed even better performance:

$$\operatorname*{Argmax}_x (\nabla f(x)/L_i)$$

Since at each iteration we need to evaluate the whole gradient and search for the best index in order to choose the block to be used in the update, the GS BCGD is costly, especially as we move towards big data applications.

4 Results of Artificial Dataset

We decided to test the Algorithms first on the dummy Dataset, trying different strategies for selecting the learning rate and choose the best 3 that will be tested on the real world dataset.

4.1 Gradient Descent. After implementing the Gradient Descent algorithm, we first tested it using a fixed (constant) learning rate. Afterwards, we analyzed other strategies for choosing the learning rate, such as the Lipschitz constant (1/L) and the Armijo Rule. For the fixed step-size, we tried several values: 0.0001, 0.0005, 0.001, 0.005 and 0.01 and analyzed the resulting curves.

The best outcomes were received with the values 0.0001 and 0.0005 for the learning rate, so we proceeded to compare them with the other learning rate strategies: 1/L and Armijo Rule. The Armijo Rule learning rate strategy reached maximum (100%) accuracy and converged in only 7 iterations. However, CPU time has to be taken into consideration as well. It was the slowest one, so we won't be proceeding doing more tests with it. The learning rate using the Lipschitz constant (1/L) showed great performance. It converged quickly, reaching the plateau of the loss function. Since this is a more unbiased choice, than using the fixed learning rate, we decided to declare this approach as the winner for the gradient descent.

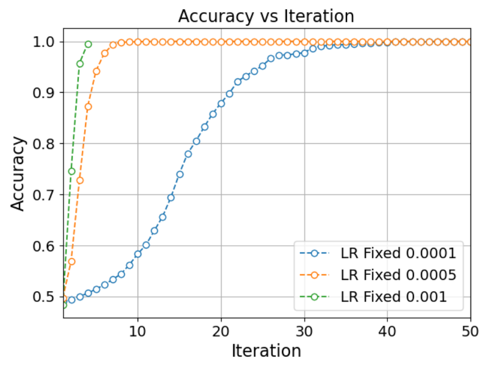


Fig. 2 **Fixed step size Gradient Descent - Accuracy vs Iterations**

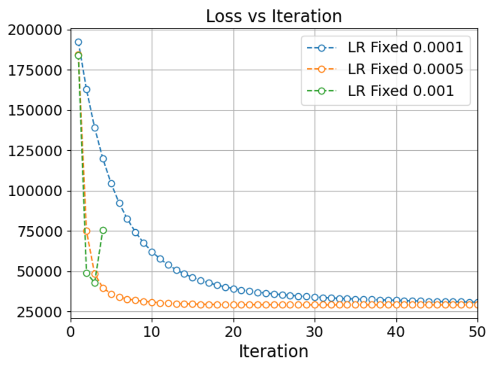


Fig. 3 **Fixed step size Gradient Descent - Loss vs Iterations**

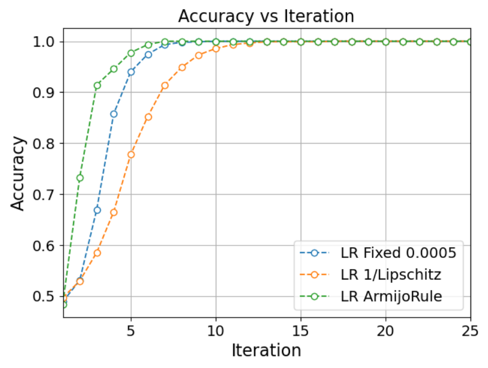
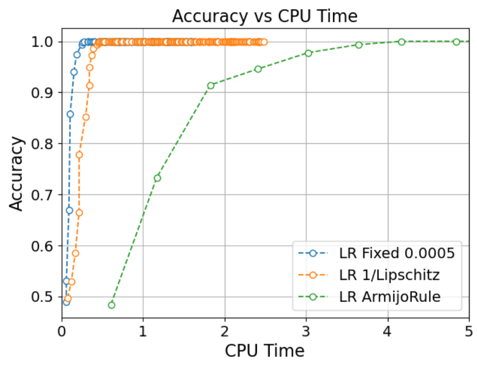
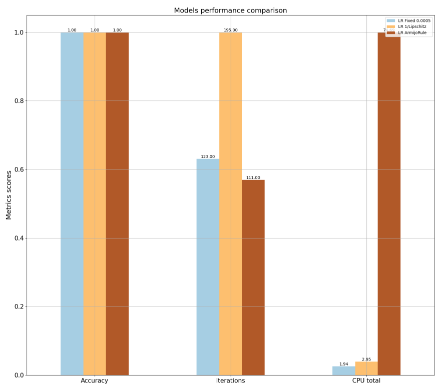


Fig. 4 **Gradient Descent Constant, Lipschitz and Armijo learning rate strategies- Accuracy vs Iterations**



(*a*)



(*b*)

Fig. 5

4.2 BCGD - Randomized rule. Again, we proceed using the same testing scheme as before: we compare different learning rate strategies. In this case, despite constant learning rate, 1/L (Lipschitz constant) and Armijo rule, we also tried 1/Li. Additionally, except using uniform probability for selecting the block, we tried the Nesterov's version, in which the probability of selecting each block is: $P(ik = i) = Li/sum(Li)$. In this way blocks with a larger Lipschitz constant are drawn more often. The constant learning rate strategy was tested with the same values as before: 0.0001, 0.0005, 0.001, 0.005, 0.01. After analysing the results, here are the conclusions:

When using learning rates of 0.01 and 0.005, the loss starts to decrease, but at some point diverges and becomes infinity. We observed a similar behavior for 0.001: the loss at first decreases, but at one point starts to increase and it diverges. The learning rate of 0.0001 worked well with decreasing the loss, but was very slow, and needs many iterations (more than 5000 for even the synthetic data).

So, by analyzing the curves, we pick 0.0005 as the best performing fixed learning rate. We proceeded to compare it with the other learning rate strategies: 1/L, Armijo Rule, 1/Li, and also the above-mentioned Nesterov sampling with 1/L and 1/Li as learning rates.

The following conclusions can be given after observing the curves for each configuration of the optimization algorithhm.

- The slowest were: Nesterov with 1/Li and Armijo
- The fastest were: constant learning rate and 1/L.
- The best accuracy and loss results: 1/Li.

Considering the values for the accuracy and CPU time, we will proceed with the last one (1/Li).

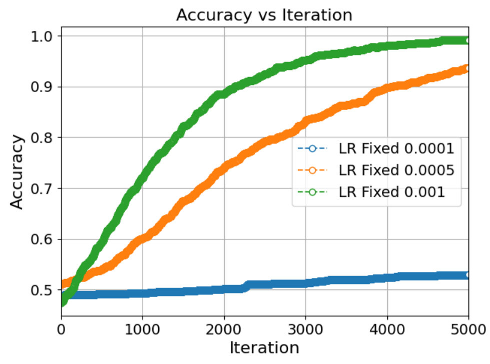


Fig. 6 RBCGD learning rate strategies - Constant learning rates, Iteration vs Accuracy

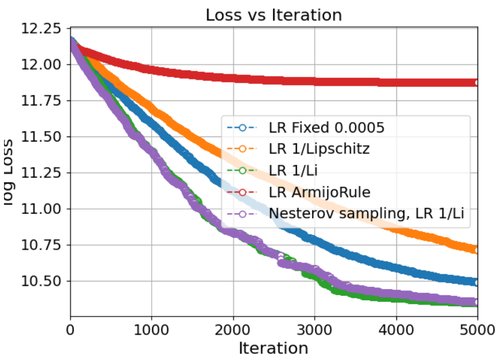


Fig. 7 RBCGD learning rate strategies - Constant, Lipschitz learning rate strategy, Armijo learning rate strategy, 1/L and Nesterov Sampling 1/Li, Loss vs Iterations

4.3 BCGD - Gauss - Southwell rule. For the Block coordinate gradient descent with Gauss-Southwell rule we tested the algorithm with different block selection methods, as described above, and different learning rate strategies, specifically: constant learning rate, 1/L, 1/Li and Armijo Rule. For the fixed step sizes we tested the values of: 0.0001, 0.0005 and 0.001. Other values outside this range either diverged, or showed no progress. We chose the one value of 0.0005 and continued to compare it with the rest.
During training, we encountered very high CPU times when using Armijo Rule, so we decided to exclude it from the analysis.
From the comparison bar chart on Fig.10 we can see that 1/Lipschitz constant and the fixed LR strategy didn't acheive maximum(100%) accuracy. All the rest performed very

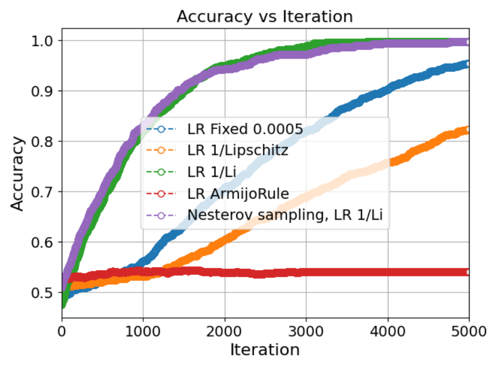
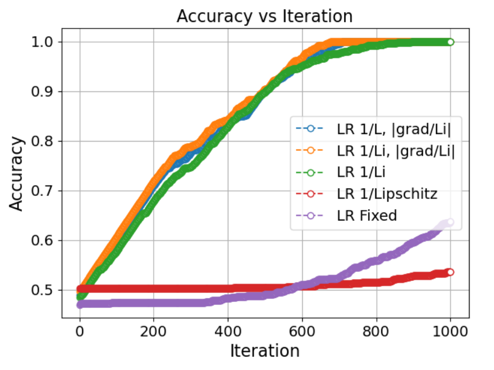
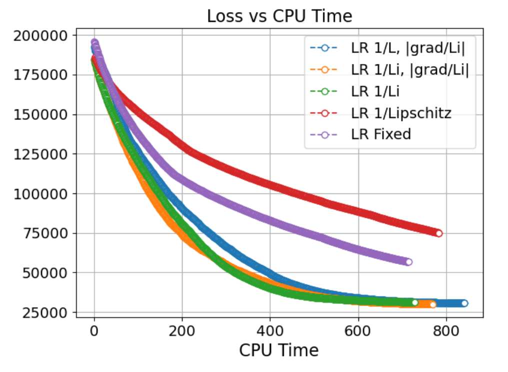


Fig. 8 RBCGD learning rate strategies - Constant, Lipschitz learning rate strategy,Armijo learning rate strategy, 1/L and Nesterov Sampling 1/Li, Iteration vs Accuracy

well. Also, the CPU times are comparable (we don't see a drastic difference), so by theory we chose the configuration with 1/Li as learning rate that utilizes $\text{Argmax}_x(\nabla f(x)/L_i)$ for choosing the block.



(*a*)



(*b*)

Fig. 9

5 Results of Publicly Available Dataset

We tested the three chosen algorithms on a **credit card defaulter data set** that we obtained from Kaggle. We decided to use the features balance and income. Because of resources limitations, we performed the tests on a subset of points, chosen randomly. The original dataset has 5776 rows, but as per the training we decided to use 1000 of them. During training we used 10% as the ratio of the labeled points, but here we proved that even having a smaller portion of labeled points works well. In our case, we chose 3% as the number of labeled points. From the comparison between the three chosen models, we can draw the following conclusions:

- Both the gradient descent and BCGD with Gauss Southwell rule managed to label all of the points correctly (100% accuracy).
- Randomized BCGD yielded 82% accuracy.
- GS BCGD needed most time to complete, which was expected. However Randomized BCGD took more time than the Gradient Descent, which was unexpected. One possible explanation would be that the calculation of the Hessian (for obtaining Li) is more costly than computing the full gradient.

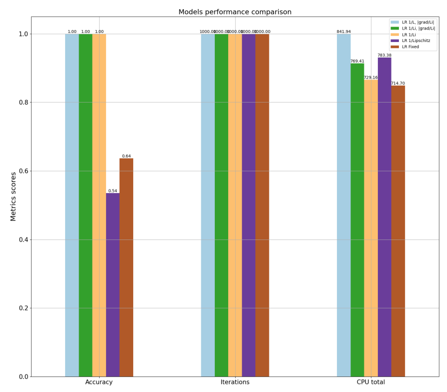


Fig. 10 GS BCGD Comparison model

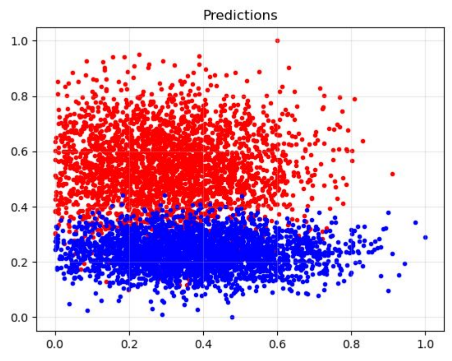


Fig. 11 Real Dataset Classification points

6 Conclusion

This paper compares the performance of gradient-based optimization algorithms, applied to solve the semi-supervised labelling problem. Even though the initial test-

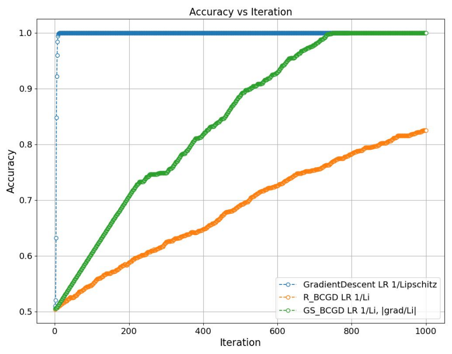


Fig. 12 GD Real Dataset- Accuracy vs Iterations

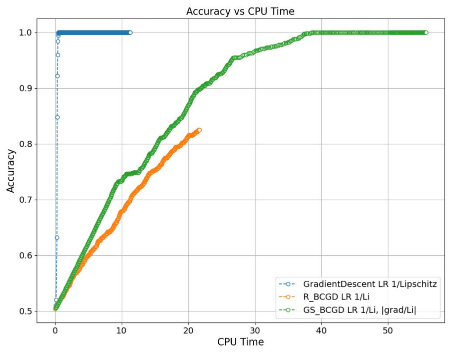


Fig. 13 GD Real Dataset- Loss vs CPU time

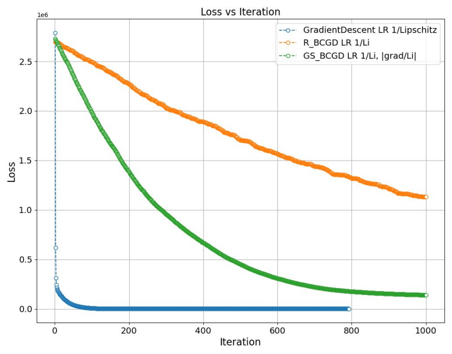


Fig. 14 RBCGD Real Dataset - Loss vs Iterations

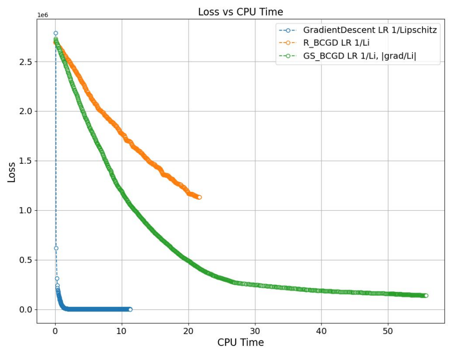


Fig. 15 RBCGD Real Dataset - Accuracy vs Iterations

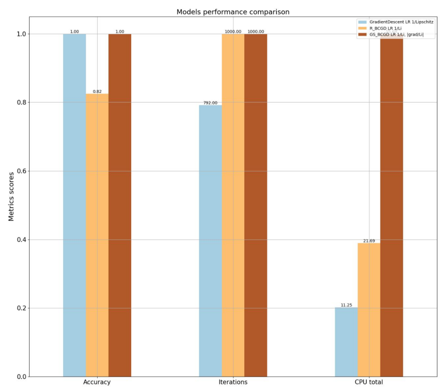


Fig. 16 Final Model Comparison

ing was done on highly separated points, the real dataset had some overlap and the models still performed decently.

List of Figures