

NOTE TO USERS

Page(s) not included in the original manuscript and are unavailable from the author or university. The manuscript was scanned as received.

iii

This reproduction is the best copy available.

UMI[®]

A

PATTERN RECOGNITION AND THE WHITEHEAD
MINIMIZATION PROBLEM

By
Alexei D. Miasnikov

A dissertation submitted to the Graduate Faculty in Computer Science in partial
fulfilment for the requirements for the degree of Doctor of Philosophy,
The City University of New York
2005

UMI Number: 3169953

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform 3169953

Copyright 2005 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

THE CITY UNIVERSITY OF NEW YORK
DEPARTMENT OF
COMPUTER SCIENCE

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

<u>5/1/05</u> Date	<u>Robert M. Hersh</u> Chair of Examining Committee
<u>5/4/05</u> Date	<u>[Signature]</u> Executive Officer

Supervision committee:

Sergei Artemov

Robert Gilman

Abstract

Pattern Recognition and The Whitehead Minimization Problem

by

Alexei D. Miasnikov

Adviser: Distinguished Professor Robert Haralick

In this research we show that the ideas of experimental mathematics and, in particular, application methods of statistical pattern recognition can be of great use in group theory.

We apply exploratory data analysis to the Whitehead Minimization Problem, which concerns itself with the problem of finding elements of the minimal length in the automorphic orbit of an element in a free group. The only known algorithm, which is due to Whitehead, has exponential dependence on the rank of a free group. Moreover, the worst case is inevitable except for some trivial cases.

We begin by describing a stochastic search procedure which surprisingly performs much better than the classical procedure. We perform statistical analysis of the solutions produced by the stochastic procedure and the non-minimal elements themselves to obtain a number of non-trivial search heuristics which allow one to predict the length-reducing automorphisms for a particular non-minimal element.

The worst case behavior of the Whitehead's procedure occurs when it is applied to a minimal element. We present a probabilistic classification system based on non-linear regression and Support Vector Machines which recognizes minimal elements with the rate of misclassification error being less than 1%.

Based on the methods described above we develop fast deterministic algorithm which significantly outperforms the original one and is able to avoid the exponential blowout on most inputs. In addition we present a probabilistic algorithm which is very robust and can be used in groups with large ranks, where any other known algorithm fails to produce.

Another important contribution of this thesis is the formulation of a number of rigorous mathematical conjectures dealing with the structure of the problem and its generic behavior (behavior on most inputs). In particular, based on statistically significant evidence we conjecture that most non-minimal elements in a free group are reducible by one of the Nielsen automorphisms which can be identified by inspecting the structure of the corresponding Whitehead Graph and that the feature vectors of minimal elements are located in a "compact" region in the corresponding space and can be bounded by a hypersurface.

Acknowledgements

I would like to thank Professor Robert Haralick, my advisor, for his many suggestions and constant support during the research and writing of this thesis. His philosophy and unique approach to pedagogy have taught me a great deal, be it in science or in everyday life.

I am very thankful to Professor Alexei G. Myasnikov, my uncle, whose unshakable enthusiasm and unwavering belief in me gave me the impetus to persevere.

I am grateful to my parents for too many things to mention, but above all, for their love. To my father for inspiring me not to give up: it took me a PhD and having child of my own to realize and appreciate what he has done.

My cherished wife, Lena, deserves far more than mere thanks. For all the times she carried the satchel of family responsibilities, for her steadfast support and gentle patience in enduring this every step of the way without uttering even a single complaint, my gratitude - my heart - is hers. Without her this work would never have come into existence (literally).

Needless to say, my sincere thanks are sent to my friends: Joseph Driscoll (for all the support I received during these years); Jayson Rome (for the frequent and seemingly endless discussions we had); Pavel Chibisov and Gabriel Zapata (for all the good – and bad – times we had together).

Finally, I gratefully acknowledge the kindnesses of my friends and colleges who (perhaps without their even knowing it) have helped me immeasurably by giving advice, inspiring me with their unimpeachable work ethic and lending an ear when the going got rough: in particular, Dmitri Bormotov, Sasha Ushakov, Professors Gilbert Baumslag, Vladimir Shpilrain, Sergei Artemov, and Robert Gilman, as well as Roman Foxman, Rave Harpaz, Stephan Aultmiller and Guangmin Shi. Spacebo.

New York

Alexei Miasnikov

April 27, 2005

Table of Contents

Acknowledgements	vi
Table of Contents	viii
List of Tables	xii
List of Figures	xvi
Introduction	1
1 Basics of Statistical Pattern recognition	8
1.1 Introduction	8
1.2 Feature extraction	10
1.3 Supervised learning	13
1.3.1 Bayesian decision rule	13
1.3.2 Regression	18
1.3.3 Fisher Discriminant	20
1.3.4 Support Vector Machines	22
1.3.5 Decision trees	27
1.4 Clustering	31
1.4.1 Hierarchical algorithms	32
1.4.2 Clustering algorithms based on function optimization	34
1.4.3 Clustering algorithms based on graph theory	40
1.5 Stochastic search techniques	42
1.5.1 The general model	42
1.5.2 No free lunch theorem	45
1.5.3 Hill Climbing	46
1.5.4 Simulated Annealing	47
1.5.5 Tabu Search	49
1.5.6 Genetic Algorithms	51

1.6	Hypothesis Testing	53
1.6.1	Pearson's χ^2 test	55
1.6.2	k -class mean comparison	56
1.6.3	Combining statistics for multivariate observations	61
1.7	System evaluation	62
1.7.1	Cross-validation and bootstrapping	62
1.7.2	Accuracy counts	64
1.7.3	Receiver Operating Characteristic	65
1.7.4	Evaluation of Clustering algorithms	67
2	The Automorphism Problem in Free Groups	70
2.1	Introduction	70
2.2	Whitehead Algorithm	76
2.2.1	Decision algorithm for RP	76
2.2.2	Decision algorithm for MSP.	77
2.3	Estimates for the time-complexity of the Whitehead's algorithms. . .	79
2.3.1	Algorithm DWR.	79
2.3.2	Algorithm DWMS	82
2.4	General Length Reduction Problem.	84
2.5	The Whitehead's complexity	88
3	Experimenting with free groups	91
3.1	Experimental Protocol	92
3.2	Data generation	96
3.2.1	Generating "random" elements in a free group	96
3.2.2	Generating "random" automorphisms of a free group	100
3.2.3	Generating "random" non-minimal elements in a free group . .	101
3.3	Minimal sample size estimation.	102
3.3.1	Estimating the minimal size of a sample set of boolean obser- vations.	102
3.3.2	Estimating the minimal size of a set of real valued observations. .	104
3.4	Testing the Consistency	106
3.4.1	Even/odd length ratio	106
3.4.2	Counting the Generators	108
3.4.3	Test based on the Whitehead Graph	108
3.4.4	Invariance under permutations of the generators	109
3.5	Experimental Comparison of the Algorithm Performance	112
3.5.1	Experimental "Generic" behaviour	113
3.5.2	Counting the Successful Runs	116

4	Stochastic Search Algorithm for the Whitehead Minimization Problem	118
4.1	Introduction	118
4.2	Description of the Algorithm	119
4.2.1	Solutions and members of the population	119
4.2.2	Evaluation method	121
4.2.3	Termination condition	121
4.2.4	Stochastic operators	123
4.2.5	Precise solutions and local search	125
4.2.6	Steps of SWR	126
4.3	Complexity Estimates and Performance evaluation	128
4.3.1	Complexity of Length Reduction Problem	128
4.3.2	Complexity Estimates of Components of SWR	132
4.3.3	Experiments with primitive elements	135
4.3.4	Comparison of the Standard Whitehead algorithm DWR with the Stochastic Whitehead algorithm SWR	137
5	Recognition of Whitehead-minimal elements	141
5.1	Introduction	141
5.2	Description of the system	142
5.2.1	Training Data Set	143
5.2.2	Features	146
5.2.3	Model	149
5.2.4	Evaluation	153
5.2.5	Feature selection algorithm	155
5.3	Experiments in F_2	157
5.3.1	Evaluating classifiers	157
5.3.2	Feature selection and analysis of the pattern recognition systems	160
5.4	Experiments with Regression in F_3 , F_4 and F_5	164
5.5	Support Vector Machine Classifier	167
6	Search Heuristics for the Whitehead Minimization Problem	178
6.1	Introduction	178
6.2	Nielsen first	179
6.3	Cluster analysis	181
6.4	Improvement on the clustering	184
6.5	Maximal weight automorphisms first	186
6.6	Comparison of the strategies	191
6.6.1	Estimates of the expected performance	191

6.6.2	Experimental results	195
7	A Hybrid Search Algorithm for the Whitehead Minimization Problem	208
7.1	Introduction	208
7.2	Description of the Hybrid algorithms	209
7.2.1	HDWR	209
7.2.2	HPWR	212
7.3	Evaluation	214
	Conclusion	222
	Bibliography	226

List of Tables

1.1	Frequencies of substrings in English text.	12
1.2	The weather data with counts.	16
1.3	A new day.	17
2.1	Estimates of LR_n/A_n on inputs of various lengths, $n = 2, 3, 4, 5$	82
2.2	$WC(w)$ vs $ w $. Average ratio $ wt / w $ for Whitehead and Restricted Whitehead automorphisms.	89
4.1	Correlation between $ w $ and T_r , $r = 2, 5, 10, 15, 20$. Correlation is negative for large words which means that the number of generations T_r does not increase with the input size.	130
4.2	Values of $E(r)$ and the confidence for $r = 2, 5, 10, 15, 20$	131
4.3	Maximal and average lengths of the population members.	135
4.4	Fraction of elements $Q_{r,1}$, $r = 2, 5, 10, 15, 20$ for which $Gen_r(w) \geq E(r) w $, where $E(r)$ is the average expected number of generations required for SWR to solve the length reduction problem. The termination condition T3 gives reliable results even with $c = 1$	136
4.5	Values of $Q_{r,1}$ and the corresponding 95% confidence intervals. . . .	137
4.6	Performance comparison of DWR and SWR. SWR significantly outperforms DWR in free groups with large ranks.	139

5.1	Description of the training and test datasets in F_2	154
5.2	Performance of the classifiers based on linear mappings in F_2	159
5.3	Performance of the classifiers based on quadratic mapping in F_2	160
5.4	Comparative results for $\mathbf{P}_*$	162
5.5	Fraction of elements in $NM(S_e)$, $NM(S_{10})$ and $NM(S_P)$ reduced by automorphisms from T_1 and T_2	173
5.6	Description of the training and test datasets in free groups F_3 , F_4 and F_5	174
5.7	Performance of the classifier $\mathbf{P}_{WG}^2$ in free groups F_3 , F_4 and F_5	175
5.8	Description of the training and test datasets in free groups F_{10} , F_{15} and F_{20}	176
5.9	Performance of the Support Vector Machine classifier in free groups F_3 , F_5 , F_{10} , F_{15} and F_{20}	177
6.1	Description of the test sets of non-minimal elements in free groups F_3 , F_4 and F_5	180
6.2	Fraction of elements in the test sets S_1 , S_{10} and S_P with length-reducing Nielsen automorphisms.	181
6.3	Average values $avg(R_{max})$ of the goodness measure R_{max} for K -means clustering of test sets S_1 , S_{10} and S_P . K is the number of clusters.	183
6.4	Average values $avg(R_{max})$ of the goodness measure R_{max} for the clustering based on the distance to the estimated centers λ_t	185
6.5	Edge weight distributions in the cluster centers λ_{t_1} and λ_{t_2} of a free group $F_3(X)$, $X = \{a, b, c\}$, where $t_1 = (a \rightarrow ab^{-1}, b \rightarrow b, c \rightarrow c)$ and $t_2 = (a \rightarrow a, b \rightarrow a^{-1}b, c \rightarrow c)$. Rows are sorted by the edge weights.	188

6.6	Values of the goodness measure $\mathcal{G}_{MAX}$ for sets of non-minimal elements in free groups F_3 , F_4 and F_5	189
6.7	Values of PCT_{99} computed for different heuristics applied to test sets S_1 and S_{10} in free groups F_3 , F_4 and F_5	194
6.8	Results of experiments with sets of primitive elements in free groups F_3, F_4 and F_5 . Counts are averaged over all inputs.	197
6.9	Correlation coefficients between a word's length and values of N_{LRP} . Negative coefficients indicate that N_{LRP} does not increase when length increases.	198
7.1	Comparison of algorithms DWR, HDWR and HPWR on the test sets of primitive elements S_P in free groups F_3 , F_4 and F_5 , where N_s is the average number of elementary steps to find a minimal element, N_{red} the average number of length-reducing steps, T_{avg} is the average time (in seconds) spent on an input and $E(\mathcal{A})$ is the error rate of $\mathcal{A}$	219
7.2	Comparison of algorithms DWR, HDWR and HPWR on the test sets S_1 in free groups F_3 , F_4 and F_5 , where N_s is the average number of elementary steps to find a minimal element, N_{red} the average number of length-reducing steps, T_{avg} is the average time (in seconds) spent on an input and $E(\mathcal{A})$ is the error rate of $\mathcal{A}$	220
7.3	Comparison of algorithms DWR, HDWR and HPWR on the test sets S_{10} in free groups F_3 , F_4 and F_5 , where N_s is the average number of elementary steps to find a minimal element, N_{red} the average number of length-reducing steps, T_{avg} is the average time (in seconds) spent on an input and $E(\mathcal{A})$ is the error rate of $\mathcal{A}$	221

7.4	Performance of the algorithm HPWR on sets of primitive elements in free groups F_{10} , F_{15} and F_{20} , where N_s is the average number of elementary steps to find a minimal element, N_{red} the average number of length-reducing steps, T_{avg} is the average time (in seconds) spent on an input and $E(\mathcal{A})$ is the error rate of $\mathcal{A}$	221
-----	--	-----

List of Figures

1.1	Posterior probabilities for two class-conditional probabilities. The shaded are represents the classification error.	16
1.2	Agglomerative hierarchical clustering algorithm combines clusters into larger ones using similarity measure between sets of points.	33
1.3	Generalized Mixture Decomposition Algorithm.	37
1.4	k -means clustering procedure.	39
1.5	Minimal Spanning Tree Clustering algorithm.	41
1.6	Hill Climbing.	48
1.7	Simulated Annealing (Kirkpatrick et al. 1983).	50
1.8	Tabu search (Glover, Hansen 1986).	51
1.9	Genetic Algorithm (Holland, 1975).	54
1.10	Test of hypothesis H_0	55
1.11	Receiver Operating Characteristic.	66
2.1	Whitehead's Algorithm for two elements u and v . Two stage procedure: computes u_{min}, v_{min} ; enumerates all elements of length $ u_{min} = v_{min} $	78
2.2	Estimates of LR_n/A_n on inputs of various lengths, $n = 2, 3, 4, 5$	82

2.3	Whitehead method for reduction problem. On each iteration a length-reducing automorphism has to be found. This procedure depends exponentially on the rank of a free group and the main obstacle in solving the problem efficiently.	85
3.1	Fragments of Cayley graphs: a) infinite abelian group on two generators $Z \times Z$; b) free group on two generators F_2 . Vertices represent elements of the corresponding group.	98
3.2	Fractions of Whitehead-minimal elements in a free group F_r , $r = 3, 4, 5$. Almost all elements of length greater than 200 are Whitehead minimal.	99
3.3	Optimal line l^* in the log space with K points above the line.	114
3.4	Examples of misrepresenting lines.	115
4.1	Algorithm SWR.	127
4.2	Values of T_r , $r = 2, 5, 10, 15, 20$. T_r stabilizes as length of the input elements grows.	129
4.3	Values of $ \mu $ for various word lengths: a) maximal $ \mu $, b) average $ \mu $. For most inputs $ \mu $ can be bounded by a constant.	134
4.4	Time comparison between standard and genetic algorithms on primitive elements in a) F_2 , b) F_5 and c) F_{10}	140
5.1	Conditional probabilities.	170
5.2	Conditional probabilities for $\mathbf{P}_*$	171
5.3	Scatter plot of points $f^*(w)$, $w \in S_e$	171
5.4	Conditional probabilities for $\mathbf{P}_*$ on S_{10}	172

6.1	Each cluster C_i in free groups F_3 , F_4 and F_5 is plotted against the corresponding transformation t_{C_i}	199
6.2	Values of estimates $\hat{P}_{NF}$ obtained from the test set S_1 in free groups F_3 , F_4 and F_5	200
6.3	Values of estimates $\hat{P}_{NF}$ obtained from the test set S_{10} in free groups F_3 , F_4 and F_5	201
6.4	Values of estimates $\hat{P}_C$ obtained from the test set S_1 in free groups F_3 , F_4 and F_5	202
6.5	Values of estimates $\hat{P}_C$ obtained from the test set S_{10} in free groups F_3 , F_4 and F_5	203
6.6	Values of estimates $\hat{P}_{MAX}$ obtained from the test set S_1 in free groups F_3 , F_4 and F_5	204
6.7	Values of estimates $\hat{P}_{MAX}$ obtained from the test set S_{10} in free groups F_3 , F_4 and F_5	205
6.8	Values of N_{LRP} for DWR_R , DWR_{NF} , DWR_C and DWR_{MAX}	206
6.9	Values of N_{red} for DWR_R , DWR_{NF} , DWR_C and DWR_{MAX}	207
7.1	Flowchart diagram of the algorithm HDWR.	217
7.2	Flowchart diagram of the algorithm HPWR.	218

Introduction

In the book, "Experimentation in Mathematics," authors J.Borwein and D.Bailey [10, 11] write: "One of the greatest ironies of the information technology revolution is that while the computer was conceived and born in the field of pure mathematics, through the genius of giants such as John von Neumann and Alan Turing, until recently this marvellous technology had only a minor impact within the field that gave it birth."

Computers have been an essential tool of any researcher working in fields ranging from applied mathematics and physics to medicine or biology. To the contrary, most mathematicians are still looking at the computers as mere calculators or printing devices, not acknowledging the fact that computer simulations and experiments could be a very useful instrument in tackling mathematical problems. In fact, experiments have always been an important tool in mathematical discovery. Carl Friedrich Gauss once declared that his way of arriving at mathematical truth was "through systematic experimentation." In recent years there was a definite shift toward this direction. Computers as well as various mathematical software packages have gained enough power to be able to perform sophisticated computations right on the office desk. If properly utilized, they might enable a completely new approach to discovery by running experiments and analyzing the outcomes.

At this point we would like to mention a few examples where computer computations were essential in obtaining mathematical results. One of the most controversial proofs accepted to a major mathematical journal in recent years was Hales' computer-assisted proof of the Kepler's conjecture [35]. The problem of finding the densest packing of spheres was introduced by Kepler in 1611. T.Hale was able to reduce the problem to one of global optimization. He solved it by systematically applying linear programming methods to 5,000 different configurations of spheres which involved solving around 100,000 linear programming problems. The proof contains 250 pages and 3 gigabytes of computer programs and results. After a year of work, a team of 12 referees concluded that they are 99% sure that the proof is correct, but they cannot verify the correctness of all computer computations. Another prominent example is the solution of the Four Colour Conjecture, where researches used more than 1,200 hours of computer time to work through the details of the final proof [2, 4, 3]. Computers were used in solving a number of key equations over free groups during the process of discovering a solution to the famous Tarski problem [40]. A computer program was responsible for discovering of a direct formula for π , which allows the computation of individual hexadecimal or binary digits of π [6]. An interesting fact is that proving that the formula is correct was almost a trivial matter after it had been obtained. Finally, we would like to mention that intensive computations preceded the proof of the famous Fermat's last theorem [17]. With the help of computers Fermat's last theorem was proved to be true for integers up to $4 \cdot 10^6$.

The process of using advanced computing technology in mathematical research is often termed as *Experimental Mathematics*. The power of the computer experiment approach lies in the belief that it can lead a mathematician to discoveries and insights

that might never have been reached using traditional methods. We have to say here, that in spite of growing interest and acceptance of the new technology among working mathematicians, there is still no common methodology. Applications are usually limited to exhaustive enumerations, visualizations or verifying the correctness of suggested proofs.

One of the main objectives of this research is to show that the techniques of statistical pattern recognition can be successfully used in abstract algebra and the theory of infinite groups in particular. The approach produces exploratory methods that are helpful in revealing hidden mathematical structures and formulating rigorous mathematical hypotheses. Our philosophy here that if an irregular or non-random behavior has been observed during an experiment then there must be a pure mathematical reason behind this phenomenon, which can be uncovered by a proper statistical analysis. The discovered knowledge can be of great interest to mathematicians. In addition, one can use the obtained knowledge to develop new (perhaps probabilistic) methods to solve hard combinatorial problems in algebra.

We apply these ideas to an analysis of the *Whitehead Minimization* problem, which is the problem of finding elements of the minimal length in the automorphic orbit of a given element in a free group. This problem is of great importance in group theory and topology and continually attracts a great deal of attention from the research community. One of the most important applications of the Whitehead Minimization problem is that its solution is a part of the solution to the famous *Automorphism Problem* in free groups introduced by J.H.C.Whitehead in 1936. Methods used to solve Whitehead Minimization problem and their generalizations are used in solving equations over free groups. The same methods can be used to decide

whether an element is a part of a generating basis of a free group. To practitioners the Whitehead Minimization problem could be of interest because of its relation to a non-commutative variation of the public key cryptographic scheme by T.T.Moh [47].

All known methods of solving the Whitehead Minimization problem have exponential dependence on the rank of a free group. Moreover, the worst case occurs when the input word is reduced to a minimal one. Since the goal of the Whitehead Minimization problem is to find a minimal element in the orbit, the worst case is inevitable for almost all elements except for elements of a very particular type. This means that the known deterministic techniques are not suitable for groups of large ranks.

Using methods of pattern recognition and exploratory data analysis we were able to show that by using proper strategies one can construct the length reduction process which can be very efficient on most inputs. We conjecture that for most elements in free groups it will require not more than a quadratic number of steps to find a length-reducing automorphism.

We introduce a probabilistic classification system (classifier) which can quickly decide whether a given word is minimal or not. This classifier can be used efficiently to avoid the exponential number of steps, unavoidable in the deterministic case, if we can accept the possibility of a small classification error.

Using empirical results we formulate several conjectures regarding the various properties of the problem. One of the most interesting conjectures connects the properties of a particular element in a free group with a corresponding length-reducing automorphism, if it exists. If the conjecture is true, it will allow one to reduce a given element by effectively finding the appropriate automorphisms.

Finally, based on these conjectures, we designed two variations of the Whitehead Reduction algorithm which solve the reduction problem in nearly optimal time.

The rest of the dissertation is organized as follows. In Chapter 1, we give a brief introduction to the subject of statistical pattern recognition. The emphasis is made on the methods and approaches used during the current research and will be referenced in the chapters that follow.

Chapter 2 gives an introduction to the Automorphism problem and the Whitehead Reduction Problem. We concentrate on the computational aspects which arise in solving these problems. Description of the famous Whitehead's algorithm and the latest results on the complexity of the two problems are given. We argue that the analysis of the complexity based on the length of the input words does not give an adequate picture when comparing the performance of different reduction algorithms. We introduce a new "size" function, called *Whitehead Complexity*, which we believe is suited better for this purposes.

In Chapter 3 we discuss the general problems that arise in experimenting with free groups. The notion of randomly generated elements in a free group are discussed and generating procedures for different types of elements are presented. We introduce statistical methods for estimating the minimal size and testing the consistency and representativeness of the generated datasets. In addition we present some techniques for experimental comparison of the performance of probabilistic and incomplete algorithms.

We begin the analysis of the Whitehead Reduction problem in Chapter 4, where we describe a stochastic reduction algorithm SWR based on the genetic approach. In spite of its simplicity, SWR performs surprisingly well - especially in groups of

large ranks. It gave us the first indication that the Whitehead Minimization problem might have a better solution at least on some inputs. The algorithm SWR is an incomplete search algorithm. One of the main problems in implementing such methods is determining robust and efficient termination conditions.

In Chapter 5 we present a probabilistic classifier which recognizes whether or not a given element is minimal. The classifier gives a fast probabilistic termination condition for an incomplete Whitehead reduction procedure. We give two variations of the classifier. The first is based on the quadratic regression, and the second uses the *Support Vector Machines* approach. Both versions are robust and efficient and give a very small error of misclassification. However, the latter approach seems to be more suitable for free groups with large ranks.

Chapter 6 describes a number of search strategies (heuristics), suggested in order to speed up the reduction process. The heuristics are based on the empirical observations about the behavior of the problem on a number of test sets of elements from various free groups. We show that heuristics, indeed, increase the efficiency of the algorithm and we formulate several conjectures about the structure of the problem and its generic behavior.

In the final chapter we present two *hybrid* algorithms which combine all of the methods introduced in the previous chapters. The first hybrid algorithm is a deterministic algorithm which has a nearly optimal reduction procedure, but still suffers from the worst case behavior occurring when the input word is reduced to a minimal. The second algorithm is a probabilistic algorithm, where the decision of whether a word is minimal or not is made using a probabilistic classification system. This approach allows one to avoid the exponential blowout in the computation for the price

of a small classification error.

In the conclusion we summarize the main results and contributions of the present research and propose several directions in which this work can be extended in the future.

Chapter 1

Basics of Statistical Pattern recognition

1.1 Introduction

The field of pattern recognition has been actively developing for several decades. It has been successfully applied in a large number of diverse fields, ranging from computer vision, speech recognition and web search to geological analysis and recognition of bubble chamber tracks in elementary particle accelerations.

The typical application of Pattern Recognition techniques is classification of a variety of given objects into categories. In Statistical Pattern Recognition objects are presented as tuples of measurements or observations, defining points in an appropriate multidimensional space. Such measurements are usually called *features* and a tuple of observations corresponding to one object is called a *feature vector*. Most of the approaches in Statistical pattern recognition are based on statistical characterizations of features, assuming that objects are generated by a probabilistic system.

Every complete pattern recognition system has to address all of the following issues:

Obtaining the data. The data can be obtained from the real world or generated by reliable procedures, which provide independent and representative sampled data.

Feature extraction. The feature extraction mechanism computes numeric or symbolic information from the observations which are contained in the acquired datasets.

Model selection. Model selection specifies the classification or description scheme that does the actual job of classifying or describing observations, relying on the extracted features. The model is the theoretical basis for the classifier. A particular choice of the model determines the basic design of the classifier (though there might be some variations in the implementation). Model selection is one of the most active areas of research in pattern recognition. Usually model selection is closely related to the feature extraction. In practice, one may try several standard models starting with the simplest ones or more economic ones.

Evaluation. Evaluation of the performance of a particular classification system is an important component of the pattern recognition task. It answers the question whether or not the given system performs as required.

Analysis of the system. Careful analysis of the performance of a particular classifier may improve feature extraction and model selection. For example, one can look for an optimal set of features or for a more effective model. Moreover, through analysis of the most significant (insignificant) features one may gain new knowledge about the original objects.

Generally, pattern recognition techniques can be divided in two principal types:

1. *supervised learning*;

2. *unsupervised learning (clustering).*

The supervised learning algorithms are based on the availability of a set of objects that have already been classified or described. This set of objects is termed the *training set*.

In unsupervised learning the system is not given an a priori labelling of objects. Instead it establishes the classes or “natural groupings” itself based on the statistical regularities of the corresponding features. The definition of classes varies from one approach to another and based on the assumptions accepted by a particular system.

In this chapter we give a brief introduction to the subject of statistical pattern recognition and its basic techniques. Emphasis is made on the methods and approaches which were used during the current research and are referenced in the chapters that follow. An extensive introduction to pattern recognition can be found in [20, 25, 46].

1.2 Feature extraction

The classification decision about an object made by a pattern recognition scheme is based on the information contained in the feature representation of that object. The task of selecting the appropriate set of features is one of the crucial steps in designing a successful pattern recognition system. By choosing a small and the most discriminative set of features we can increase not only the classification accuracy of a system but its computational efficiency as well.

There is no strict definition of a feature. The choice of features is always problem specific and often requires expert’s knowledge about the problem. The very general

notion of a feature representation is given by the following

Definition 1.2.1. *Features represent an object by measurements whose values are very similar for objects in the same category, and very different for objects in different categories.*

We have to add here that the optimal feature representation will have the smallest set of easily extracted measurements which give the smallest classification error. Different pattern recognition approaches might require different feature representations even if they yield comparable performance.

Feature extraction is a very active area of research. Below we give examples of features used in the two most common application areas of statistical pattern recognition.

Computer Vision and Image Recognition. One of the oldest and most active areas of pattern recognition research. In image recognition one has to solve the task of assigning a given image to the class of images displaying similar objects. Representative features for images have to be invariant under different transformations including translation rotation and scaling. One of the simplest examples of features used to describe images are color intensity histograms, edge information, components analysis, features extracted with mathematical morphology operators and so on.

Document Analysis. A text document written in some language contains information specific for this particular document or the class of documents to which it belongs. For example it is well known that a text written in English language will have regularities governed by frequencies of “bi-grams” and “tri-grams” as

given in Table 1.1. Such information is commonly used in cryptanalysis and information retrieval.

Bi-/tri-grams	%	Bi-/tr-grams	%
“TH”	3.0	“ED”	1.3
“HE”	3.0	“THE”	2.0
“IN”	1.9	“ING”	.8
“ER”	1.9	“AND”	.7
“AN”	1.4	“EEE”	0.0

Table 1.1: Frequencies of substrings in English text.

Regular text documents usually categorized accordingly with the frequencies of occurrences of particular words or their roots in the text. One can construct a vocabulary of roots to distinguish documents of different classes. The relative counts of occurrences of words with roots from the vocabulary can form a set of features describing each document.

The examples above show that the feature selection task is very specific and based on the expert knowledge or empirical observations about the problem. Most of the time we cannot say in advance which set of features will guarantee reliable results. A common approach in selecting the near optimal feature set is to consider as many features as possible and then try to extract the most “significant” ones using statistical methods. This process is called the *feature selection* and usually employs dimensionality reduction techniques.

Let $D = [\mathbf{x}_1, \dots, \mathbf{x}_N]$, $x_i \in \mathbb{R}^d$ be a matrix with N d -dimensional feature vectors as columns. Define their sample mean μ by

$$\mu = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i$$

and their sample covariance Σ by

$$\Sigma = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)'$$

The Principle Component Analysis applies a linear mapping to reduce the dimensionality of the original feature space. This technique specific transformation preserves the most variation in the data.

Let $\mathbf{t}_1, \dots, \mathbf{t}_d$ be the eigenvectors of Σ with corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d$. Notice that the largest eigenvalue λ_1 corresponds to the principal variate and the eigenvector $\mathbf{t}_1$ gives the direction of the largest variance.

Let θ be a threshold close to zero and $\lambda_1, \dots, \lambda_k$ be the k largest eigenvalues such that $\lambda_i \geq \theta, i = 1, \dots, k$. Often there will be only few large eigenvalues. This implies that k is the dimensionality of the subspace governing the properties of the data and the remaining $d - k$ dimensions correspond to noise.

We construct a new set represented by principle components

$$\hat{D} = \{\hat{\mathbf{x}} \mid \hat{\mathbf{x}} = T'(\mathbf{x} - \mu), \forall \mathbf{x} \in D\},$$

where T is a $d \times k$ matrix whose columns are eigenvectors $t_1, \dots, t_k$. The new feature vectors have the reduced size k which depends on the parameter θ .

1.3 Supervised learning

1.3.1 Bayesian decision rule

The Bayesian approach is a fundamental approach in statistical learning. It makes an assumption that the decision problem is posed in probabilistic terms and all of the relevant probabilistic values are known.

Given a classification task of M classes $Q = \{q_1, \dots, q_M\}$ and an object o , the Bayesian decision will assign o to the class q which has the greatest probability

$$Prob(class = q \mid object = o).$$

Suppose object can be described by a vector of measurements $\mathbf{x}$ and can be associated with one of the "states" $q_1, \dots, q_M$. The true state C is hard to predict and we consider C to be a variable, described probabilistically. Different objects will yield different measurements whose variability, again, is expressed in probabilistic terms and D is the corresponding random variable.

Given the prior knowledge about classes $P(C = q_i)$ and a data measurement $\mathbf{x}$, we want to find *posterior* probabilities $P(C = q_i \mid D = \mathbf{x})$. Recall the Bayes formula from the elementary probability theory:

$$P(C, D) = P(C \mid D)P(D) = P(D \mid C)P(C),$$

therefore

$$P(C \mid D) = \frac{P(D \mid C)P(C)}{P(D)}.$$

The probability of a measurement being $\mathbf{x}$, in case of classes Q is

$$P(D = \mathbf{x}) = \sum_{q_i \in Q} P(D = \mathbf{x} \mid C = q_i)P(C = q_i)$$

Therefore,

$$P(C = q_i \mid D = \mathbf{x}) = \frac{P(D = \mathbf{x} \mid C = q_i)P(C = q_i)}{\sum_{g_i \in Q} P(D = \mathbf{x} \mid C = g_i)P(C = g_i)} \quad (1.3.1)$$

or, informally,

$$posterior = \frac{likelihood \times prior}{evidence}.$$

The formula shows that we convert the prior probability $P(C)$ to the posterior probability $P(C | D)$, which is the probability of the class being q_i given that the measurements $\mathbf{x}$ have been observed.

The conditional probabilities $P(D | C)$ are estimated from the training dataset which records the class each measurement came from and the priors $P(C)$ usually taken to be uniform unless some expert knowledge is available.

Notice that the sum in the denominator of Equation 1.3.1 remains constant for every class q_i . The Bayes decision rule says:

$$\text{Decide } \mathbf{x} \text{ is in } q_i \text{ if } P(C = q_i | D = \mathbf{x}) > P(C = q_j | D = \mathbf{x}), \forall j \neq i.$$

It can be shown that Bayesian classifier is optimal with respect to minimizing the classification error probability. If we observe a particular $\mathbf{x}$, the error probability is

$$P(\text{error} | D = \mathbf{x}) = \sum_{q_j \in Q - \{q_i\}} P(C = q_j | D = \mathbf{x}), \text{ if we decide } q_i.$$

Clearly deciding on the class q_j with maximal class-conditional probability $P(C | D)$ will minimize the error probability for a given observation $\mathbf{x}$. The total probability of error is given by

$$P(\text{error}) = \int P(\text{error}, D) dD = \int P(\text{error} | D) P(D) dD.$$

If we ensure that $P(\text{error} | D)$ is as small as possible for all D , then the integral will be as small as possible. Thus the Bayesian decision rule is justified.

The classification error probability is not always the best criterion to be adopted for minimization. Sometimes we want assign different importance values to different errors. For example, the error of false alarm triggered by an antimissile system is not as crucial as error of mis-detection of approaching missile.

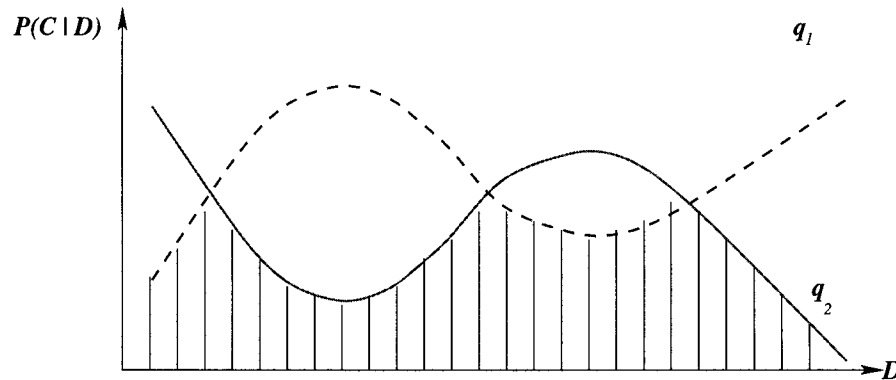


Figure 1.1: Posterior probabilities for two class-conditional probabilities. The shaded area represents the classification error.

Outlook(O)			Temperature(T)			Humidity(H)			Windy(W)			Play(Pl)	
	yes	no		yes	no		yes	no		yes	no	yes	no
sunny	2	3	hot	2	2	high	3	4	false	6	2	9	5
overcast	4	0	mild	4	2	normal	6	1	true	3	3		
rainy	3	2	cool	3	1								

Table 1.2: The weather data with counts.

A loss λ_{ki} is a penalty for misclassifying data $\mathbf{x}$ in q_i when it belongs to q_k .

Then the decision rule which minimizes average risk is:

Decide $\mathbf{x}$ is in q_i if

$$l_i \equiv \sum_{k=1}^Q \lambda_{ki} P(D = \mathbf{x} | C = q_k) P(C = q_k) < l_j \equiv \sum_{k=1}^Q \lambda_{kj} P(D = \mathbf{x} | C = q_k) P(C = q_k), \forall j \neq i.$$

Let's consider an example of the simplest Bayesian classifier (called "Naive" or "Idiot" classifier) which assumes independence between data features.

Consider the data is given in the Table 1.3.1. The joint distribution:

$$P(Pl, O, T, H, W) = P(O, T, H, W | Pl) P(Pl).$$

Now consider the task which is to decide whether a play will take place, given the

conditions described in Table 1.3.1

Outlook(O)	Temperature(T)	Humidity(H)	Windy(W)	Play(P)
sunny	col	high	true	?

Table 1.3: A new day.

From Bayes formula:

$$P(Pl|O, T, H, W) = \frac{P(O|Pl)P(T|Pl)P(H|Pl)P(W|Pl)P(Pl)}{P(O, T, H, W)} \quad (1.3.2)$$

Probabilities computed from Table 1.3.1:

$$\begin{aligned} P(Pl = yes) &= 9/14, & P(Pl = no) &= 5/14; \\ P(O = sunny|yes) &= 2/9, & P(T = cool|yes) &= 3/9; \\ P(H = high|yes) &= 3/9, & P(W = true|yes) &= 3/9; \\ P(O = sunny|no) &= 3/5, & P(T = cool|no) &= 1/5; \\ P(H = high|no) &= 4/5, & P(W = true|no) &= 3/5. \end{aligned}$$

By substituting values of marginal probabilities into the formula (1.3.2) we will obtain the following probabilities:

$$P(yes|sunny, cool, high, true) = 20.5\%$$

$$P(no|sunny, cool, high, true) = 79.5\%$$

Therefore by using Bayes decision rule we will decide “no”.

Despite the very strong independence assumption and simplicity of the “naive” classifier it often works well in practice.

1.3.2 Regression

Let $\mathbf{x}_1, \dots, \mathbf{x}_N$ be the training set of feature vectors and $y_1, \dots, y_N$ be the corresponding set of labels, where $y_i \in \{-1, +1\}$.

The classification algorithm has to predict the value $\hat{y}(\mathbf{x})$ of the class label for a new feature vector $\mathbf{x}$. One of the approaches is to explore the relationship between labels y_i and the corresponding feature vectors $\mathbf{x}_i$. Linear regression approximates the value of $\hat{y}$ via a linear function on $\mathbf{x}$:

$$\hat{y} \approx \beta' \cdot \mathbf{x},$$

where β is an unknown column vector of coefficients, usually called the regression coefficients. We form the standard regression model as

$$Y = D\beta + \epsilon,$$

where $Y = y_1 \dots y_N$ is the column vector of the known labels, D is the matrix

$$D = [\mathbf{x}_1 \dots \mathbf{x}_N]$$

with the feature vectors as rows, β is a vector of unknown regression coefficients, and ϵ represents the approximation error. Using the least squares method we find β such that the mean square error

$$\|Y - D\beta\|^2 = \|\epsilon\|^2$$

is as minimal as possible.

Taking the partial derivative

$$\frac{\partial}{\partial \beta} \|Y - D\beta\|^2 = -2D'(Y - D\beta). \quad (1.3.3)$$

and setting (1.3.3) to be equal to zero we obtain

$$-2D'(Y - D\beta) = 0$$

$$D'Y = D'D\beta$$

$$\beta = (D'D)^{-1}D'Y.$$

provided the matrix D is invertible.

Now, for a given vector $\mathbf{x}$ and the computed vector β one can obtain the label value $\hat{y}$ predicted by the regression model as

$$\hat{y} = \beta' \cdot \mathbf{x}.$$

The simple decision rule is

Decide $\mathbf{x}$ belongs to class "+1" if $\hat{y} \geq \theta$ and class "-1" otherwise,

where θ is a threshold. The natural choice for the threshold is $\theta = 0$. In some cases, careful analysis of the distribution of values $\hat{y}$ can give more accurate estimates for the threshold θ .

Unfortunately, we cannot guarantee that labels y , indeed, have a linear relationship with feature vectors $\mathbf{x}$. If the standard linear regression model fails, one can change the representation of the data replacing the feature vector $\mathbf{x}$ by its image via a non-linear mapping $\varphi = (\varphi_1, \dots, \varphi_k)$:

$$\mathbf{x} \rightarrow \varphi(\mathbf{x}) = \langle \varphi_1(\mathbf{x}), \dots, \varphi_k(\mathbf{x}) \rangle,$$

and do classification in the space induced by mapping φ . The drawback is that with many useful mappings the number of unknown coefficients β grows large.

Packages for computing linear regression models are now standard and available in many software distributions, for example, MATLAB [45], The GNU Scientific Library [33].

1.3.3 Fisher Discriminant

Let $D = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, $\mathbf{x} \in \mathbb{R}^d$ be a set of observations with corresponding labels $y_1, \dots, y_N$, $y_i \in \{-1, 1\}$.

Linear two-class classification is often performed using a linear discriminant function

$$f(\mathbf{x}) = \mathbf{w}' \cdot \mathbf{x} + b = \sum_{i=1}^d w_i x_i + b.$$

The decision rule is

Decide $\mathbf{x}$ belongs to class "+1" if $f(\mathbf{x}) \geq 0$ and class "-1" otherwise,

Fisher's linear discriminant function is obtained by finding the hyperplane $(\mathbf{w}, b)$ on which the projection of the data is maximally separated.

Let $\mathcal{C}_- = \{\mathbf{x}_i \mid y_i = -1\}$ and $\mathcal{C}_+ = \{\mathbf{x}_i \mid y_i = +1\}$ be the sets of class points. Define class sample mean

$$\mathbf{m}_c = \frac{1}{|\mathcal{C}_c|} \sum_{\mathbf{x} \in \mathcal{C}_c} \mathbf{x}.$$

The optimal hyperplane is found by maximizing the Fisher's discriminant ratio

$$F = \frac{(m_- - m_+)^2}{s_-^2 + s_+^2},$$

where

$$m_c = \frac{1}{|\mathcal{C}_c|} \sum_{\mathbf{x} \in \mathcal{C}_c} \mathbf{w}' \cdot \mathbf{x} = \mathbf{w}' \mathbf{m}_c$$

is the sample mean of the projected points from $\mathcal{C}_c$ and

$$s_c^2 = \sum_{\mathbf{x} \in \mathcal{C}_c} (\mathbf{w}' \cdot \mathbf{x} - m_c)^2.$$

is the scatter of the projected data.

To obtain F as an explicit function of $\mathbf{w}$, we define the scatter matrices S_c and S_W

$$S_c = \sum_{\mathbf{x} \in \mathcal{C}_c} (\mathbf{x} - \mathbf{m}_c)(\mathbf{x} - \mathbf{m}_c)'$$

and

$$S_W = S_- + S_+.$$

The expression for the projection scatter can be rewritten as

$$\begin{aligned} s_c^2 &= \sum_{\mathbf{x} \in \mathcal{C}_c} (\mathbf{w}' \cdot \mathbf{x} - \mathbf{w}' \cdot \mathbf{m}_c)^2 \\ &= \sum_{\mathbf{x} \in \mathcal{C}_c} \mathbf{w}' (\mathbf{x} - \mathbf{m}_c)(\mathbf{x} - \mathbf{m}_c)' \mathbf{w} \\ &= \mathbf{w}' S_c \mathbf{w}. \end{aligned}$$

The sum of scatters can be written

$$s_-^2 + s_+^2 = \mathbf{w}' S_W \mathbf{w}.$$

The separation of the projected means

$$\begin{aligned} (m_- - m_+)^2 &= (\mathbf{w}' \mathbf{m}_- - \mathbf{w}' \mathbf{m}_+)^2 \\ &= \mathbf{w}' (\mathbf{m}_- - \mathbf{m}_+) (\mathbf{m}_- - \mathbf{m}_+)' \mathbf{w} \\ &= \mathbf{w}' S_B \mathbf{w}, \end{aligned}$$

where

$$S_B = (\mathbf{m}_- - \mathbf{m}_+) (\mathbf{m}_- - \mathbf{m}_+)'.$$

Matrices S_W and S_B are called *within class scatter matrix* and *between class scatter matrix* respectively.

In terms of S_w and S_B , the Fisher's ratio can be written

$$F(\mathbf{w}) = \frac{\mathbf{w}' S_B \mathbf{w}}{\mathbf{w}' S_W \mathbf{w}}.$$

The optimum direction $\mathbf{w}$ can be found by taking the derivative of $F(\mathbf{w})$ with respect to $\mathbf{w}$ and setting it to zero:

$$\frac{\partial F(\mathbf{w})}{\partial \mathbf{w}} = (\mathbf{w}' S_W \mathbf{w})^{-2} (2 S_B \mathbf{w} \mathbf{w}' S_W \mathbf{w} - 2 \mathbf{w}' S_B \mathbf{w} S_W \mathbf{w}) = 0.$$

It follows that

$$\mathbf{w}' S_B \mathbf{w} S_W \mathbf{w} = S_B \mathbf{w} \mathbf{w}' S_W \mathbf{w}.$$

If we divide both sides by the quadratic term $\mathbf{w}' S_B \mathbf{w}$, then

$$\begin{aligned} S_W \mathbf{w} &= \frac{\mathbf{w}' S_W \mathbf{w}}{\mathbf{w}' S_B \mathbf{w}} S_B \mathbf{w} \\ &= \frac{\mathbf{w}' S_W \mathbf{w}}{\mathbf{w}' S_B \mathbf{w}} (\mathbf{m}_- - \mathbf{m}_+) (\mathbf{m}_- - \mathbf{m}_+)' \mathbf{w} \\ &= \lambda (\mathbf{m}_- - \mathbf{m}_+) \end{aligned}$$

where λ is a scalar value defined as

$$\lambda = \frac{\mathbf{w}' S_W \mathbf{w}}{\mathbf{w}' S_B \mathbf{w}} (\mathbf{m}_- - \mathbf{m}_+)' \mathbf{w}.$$

The scalar factor does not change the direction of $\mathbf{w}$ and, therefore, we can write the solution for $\mathbf{w}$ which optimizes $F(\mathbf{w})$

$$\mathbf{w} = S_W^{-1} (\mathbf{m}_- - \mathbf{m}_+).$$

The discriminant function is defined by $g(x) = \mathbf{w}' \cdot \mathbf{x}$. The vector $\mathbf{x}$ is assigned to class “+1” if $g(x) \geq \theta$ and to class “-1” otherwise. The threshold θ is set to a value that minimizes the error of the class assignment.

1.3.4 Support Vector Machines

As always we consider a set of N observations $D = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, $\mathbf{x} \in \mathbb{R}^d$ with the corresponding labels $y_1, \dots, y_N$, $y_i \in \{-1, +1\}$.

Definition 1.3.1. *The margin of an example $(\mathbf{x}_i, y_i)$ with respect to a hyperplane $(\mathbf{w}, b)$ defined as the quantity*

$$\gamma_i = y_i(\mathbf{w}' \cdot \mathbf{x} + b).$$

Note that $\gamma_i > 0$ corresponds to the correct classification of $(\mathbf{x}_i, y_i)$.

Let γ_+ (γ_-) be the smallest margin among all positive (negative) points. Define the margin of separation

$$\gamma = \gamma_+ + \gamma_-.$$

A Support Vector Machine (SVM) is a statistical classifier that attempts to construct a decision hyperplane $(\mathbf{w}, b)$ in such a way that the margin of separation γ between positive and negative examples is maximized [63, 64].

We wish to find a hyper-plane which will separate the two classes such that all points on one side of the hyper-plane will be labelled +1, all points on the other side will be labelled -1:

$$\mathbf{w}' \cdot \mathbf{x}_i + b \geq +1, \quad \text{for } y_i = +1$$

$$\mathbf{w}' \cdot \mathbf{x}_i + b \leq -1, \quad \text{for } y_i = -1$$

or equivalently

$$y_i (\mathbf{w}' \cdot \mathbf{x}_i + b) - 1 \geq 0, \quad \forall i.$$

Define a discriminant function

$$g(\mathbf{x}) = \mathbf{w}^{*'} \cdot \mathbf{x} + b^*,$$

where $\mathbf{w}^*, b^*$ are the parameters of the optimal hyper-plane. Function $g(\mathbf{x})$ gives the distance from $\mathbf{x}$ to the optimal hyper-plane. Note that function $g(\mathbf{x})$ does not change if we re-scale hyperplane to $(\lambda \mathbf{w}, \lambda \mathbf{x})$, $\lambda \in \mathbb{R}^+$, however the corresponding margin changes. Without loss of the generality we can assume that the optimal hyperplane corresponds to the one which lies within the distance 1 from the closest positive and negative points. To obtain the geometrical margin, which corresponds to the Euclidean distance to the hyperplane, we normalize weight vector $\mathbf{w}$ by $\|\mathbf{w}\|^2$. Now, let $\mathbf{x}_+$ and $\mathbf{x}_-$ be the positive and negative points with corresponding margins equal to +1 and -1 respectively. The corresponding geometrical separation margin of the resulting classifier

$$\begin{aligned}\gamma &= \frac{1}{2} \left(\frac{\mathbf{w}'}{\|\mathbf{w}\|^2} \cdot \mathbf{x}_+ - \frac{\mathbf{w}'}{\|\mathbf{w}\|^2} \cdot \mathbf{x}_- \right) \\ &= \frac{1}{2\|\mathbf{w}\|^2} (\mathbf{w}' \cdot \mathbf{x}_+ - \mathbf{w}' \cdot \mathbf{x}_-) \\ &= \frac{1}{\|\mathbf{w}\|^2}.\end{aligned}$$

We can maximize the margin by minimizing the cost function

$$\Phi(\mathbf{w}) = \|\mathbf{w}\|^2 = \mathbf{w}' \cdot \mathbf{w},$$

subject to the constraint that

$$y_i (\mathbf{w}' \cdot \mathbf{x}_i + b) - 1 \geq 0, \quad i = 1, \dots, N.$$

This is an optimization problem with inequality constraints and can be solved by means of Lagrange multipliers. We form the Lagrangian

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}' \cdot \mathbf{w} - \sum_{i=1}^N \alpha_i [y_i (\mathbf{w}' \cdot \mathbf{x}_i + b) - 1],$$

where $\alpha_i \geq 0$ are the Lagrange multipliers. We need to minimize $L(\mathbf{w}, b, \alpha)$ with respect to $\mathbf{w}$, b while requiring that derivatives of $L(\mathbf{w}, b, \alpha)$ with respect to all the α_i vanish, subject to the constraint that $\alpha_i \geq 0$. Equivalently, we can solve the dual problem of maximizing $L(\mathbf{w}, b, \alpha)$, subject to the constraints that the gradient of $L(\mathbf{w}, b, \alpha)$ with respect to $\mathbf{w}$ and b vanish, subject to the constraint that $\alpha_i \geq 0$:

$$\begin{aligned}\frac{\partial L(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^N y_i \alpha_i \mathbf{x}_i = \mathbf{0}, \\ \frac{\partial L(\mathbf{w}, b, \alpha)}{\partial b} &= \sum_{i=1}^N y_i \alpha_i = 0.\end{aligned}$$

This gives:

$$\begin{aligned}w &= \sum_{i=1}^N y_i \alpha_i x_i, \\ 0 &= \sum_{i=1}^N y_i \alpha_i.\end{aligned}$$

Substituting these constraints back into the Lagrangian we obtain:

$$L(\mathbf{w}, b, \alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j \mathbf{x}'_i \cdot \mathbf{x}_j. \quad (1.3.4)$$

Suppose parameters α_i^* is the solution to optimization problem which maximizes (1.3.4) with constraints $\sum_{i=1}^N y_i \alpha_i = 0$ and $\alpha_i \geq 0$. Then the optimal hyperplane which realizes the maximal separation γ is defined by vector

$$\mathbf{w}^* = \sum_{i=1}^N y_i \alpha_i^* \mathbf{x}_i.$$

The small subset of vectors whose corresponding Lagrange multipliers are not zero after the optimization are called the support vectors. The support vectors lie closest to the decision boundary, making them the critical elements of the training set [36].

Let SV denote the set of indexes with zero Lagrange multipliers:

$$SV = \{i \mid \alpha_i = 0\}.$$

The discriminant function

$$\begin{aligned} g(x) &= \sum_{i=1}^N y_i \alpha_i^* \mathbf{x}_i' \cdot \mathbf{x} + b^* \\ &= \sum_{i \in SV} y_i \alpha_i^* \mathbf{x}_i' \cdot \mathbf{x} + b^*, \end{aligned}$$

where α_i^* , b^* are the parameters of the optimal decision hyperplane. It shows that the distance can be computed as a weighted sum of the training data and the Lagrange multipliers, and that the training vectors $\mathbf{x}_i$ are only used in inner products.

The decision rule is similar to the previous one

Decide $\mathbf{x}$ belongs to class "+1" if $g(\mathbf{x}) \geq 0$ and class "-1" otherwise.

We can extend linear case to non-linearly separable data by introducing a kernel function

$$K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j),$$

where $\varphi(\mathbf{x})$ is some non-linear mapping into (possibly infinite) space H ,

$$\varphi : \mathbb{R}^n \mapsto H.$$

Since Support Vector Machines use only inner products to compute the discriminant function, given kernel $K(x_i, x_j)$, we can train a SVM without ever having to know $\varphi(x)$ [36]. The implication of this is that the number of parameters that has to be learned by the SVM does not depend on the choice of the kernel and, therefore, mapping φ .

This gives an obvious computational advantage when mapping the original feature space into a higher dimensional space, polynomial for example.

Examples of typical kernel functions are:

- Linear:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i' \cdot \mathbf{x}_j$$

- Polynomial:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i' \cdot \mathbf{x}_j)^d$$

- Exponential:

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}$$

- Neural Networks:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\theta_1 \mathbf{x}_i' \cdot \mathbf{x}_j - \theta_2)$$

1.3.5 Decision trees

The classifiers described above were primarily designed for a two-class classification problem where the classes are assumed to be separated by a known hypersurface. Decision trees give a nice and relatively simple way of constructing more general classifiers, including multi-class classifiers with no explicit separation assumptions [13, 15, 55].

The most commonly used type of decision trees is based on binary trees, i.e. every nonterminal node has exactly two children. Suppose we have an M -class classification problem with classes $Q = \{q_1, \dots, q_M\}$. Let

$$r(\mathbf{x}, \theta) \in \{0, 1\}$$

be a decision rule defined on feature vectors $\mathbf{x} \in \mathbb{R}^d$ and depending on a set of parameters θ . A general binary decision tree is a binary tree in which every non-leaf node n is assigned a corresponding decision rule r_n . The children of non-leaf node correspond to the two possible outcomes produced by the decision rule. Leaf nodes are usually labelled with the resulting class label or contain a probabilistic description of the form

$$Prob(class = q_i \mid leafnode), i = 1, \dots, M.$$

To classify an instance $\mathbf{x}$ using decision tree, one starts at the root of the tree. If the node's decision rule results in "1" the left child is taken; if it is "0" the right child is taken. Taking appropriate branches according to the decisions made by the node rules, we eventually will come to a leaf node. Depending on the scheme, the instance $\mathbf{x}$ is assigned to a class whose label is stored at the leaf node or whose conditional probability $Prob(class \mid leafnode)$ is maximal.

To construct(train) a decision tree we need a training set $D = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, $\mathbf{x} \in \mathbb{R}^d$ with the corresponding true class labels $Y = \{y_1, \dots, y_N\}$, $y_i \in Q$ and the set of possible decision rules $\mathcal{R} = \{r_1, \dots, r_N\}$, $r_i \in \{0, 1\}$. Note, that in general rules r_i do not have to have the same parameter values. Moreover, rules can be of completely different kinds. In fact with simple modifications, all of the classification rules described previously can be a part of the set $\mathcal{R}$.

We start at the root of the tree and repeatedly bi-partition the set D at each node. The decision rule $r_n \in \mathcal{R}$, used to partition the set at node n , is chosen to optimize the splitting evaluation criterion. The node expansion stops when an appropriate stopping condition is met.

Now we describe a rule selection procedure based on the *information gain criterion*

at the node n . Denote by $D_n \subseteq D$ a subset of the original training set which is associated with the node n . Let $N_L(r)$ and $N_R(r)$ be the number of vectors assigned by rule r to the left and the right children of n respectively

$$N_L(r) = |\{\mathbf{x} \mid r(\mathbf{x}) = 1, \mathbf{x} \in D_n\}|,$$

$$N_R(r) = |\{\mathbf{x} \mid r(\mathbf{x}) = 0, \mathbf{x} \in D_n\}|.$$

By $N_{L_c}(r)$ and $N_{R_c}(r)$ we denote the number of points assigned by r to the left and right children and whose true class is c :

$$N_{L_c}(r) = |\{\mathbf{x}_i \mid r(\mathbf{x}) = 1 \text{ and } y_i = c\}|,$$

$$N_{R_c}(r) = |\{\mathbf{x}_i \mid r(\mathbf{x}) = 0 \text{ and } y_i = c\}|,$$

where $\mathbf{x}_i \in D_n$.

The information gain criterion is based on the entropy purity function and for a given rule r at the node n is defined as

$$E_n(r) = \sum_{c \in Q} (N_{L_c}(r) \ln p_{L_c}(r) + N_{R_c}(r) \ln p_{R_c}(r)),$$

where

$$p_{L_c}(r) = \frac{N_{L_c}(r)}{N_L(r)},$$

$$p_{R_c}(r) = \frac{N_{R_c}(r)}{N_R(r)}.$$

The entropy based function gives the maximal value when the classes of the training vectors are completely separated, i.e. only instances which belong to the same

class are assigned to each of the children nodes. The decision rule r_n which maximizes $E_n(r)$ is chosen as the splitting rule at the node n .

To define the stopping criterion for the node expanding process the following conditions usually are taken into the consideration. First, the size of the tree has to be reasonably bounded. We set the maximal depth of the decision tree to be $\log_2 |D| - 1$, which guarantees that the number of leaf nodes is less than $|D|/2$. Second, if the partitioning of the data at the node n does not result in significantly different class distributions at the resulted children nodes, the further splitting could be stopped. To define the condition, when the distribution of classes in left and right children does not differ significantly we perform a statistical test, bases on the χ^2 statistic

$$\chi^2 = \sum_{c \in Q} \left(N_{L_c}(r^*) \ln p_{L_c}(r^*) + N_{R_c}(r^*) \ln p_{R_c}(r^*) - |D_n(c)| \ln \frac{|D_n(c)|}{|D_n|} \right),$$

where $D_n(c)$ is the set of points from D_n with the true labels equal to c . This statistic has a χ^2 distribution with $M - 1$ degrees of freedom. If the value of χ^2 is small, the distributions of classes in children nodes are not significantly different and we can stop the splitting process. The last condition relates to the size of the set D_n . If D_n is small χ^2 cannot give a reliable result. Moreover, often decision rules themselves put restrictions on the minimal size of the training set. We stop splitting at the node n if D_n is smaller than the predefined lower limit. If one of the conditions above is detected at node n , we stop the expanding process and n becomes a leaf node.

Every leaf node is assigned a class label, corresponding to the class which has the highest number of class labels at the training set corresponding to that node. To be

mode formal, leaf node l is assigned class c , where

$$|D_l(c)| = \max_{q \in Q} \{|D_l(q)|\}.$$

In addition, one can compute a class sampling distribution $P(\mathcal{C} \mid l)$ at the leaf node l , where

$$P(\mathcal{C} = c \mid l) = \frac{|D_l(c)|}{|D_l|}.$$

This information is often useful if inferences based on the classification procedure are probabilistic in nature.

1.4 Clustering

Unsupervised learning or *clustering* methods of pattern recognition are used when no a priori information about the objects' groupings is available. In this case algorithms establish classes or "natural groupings" themselves based on the specified similarity assumptions. Feature vectors combined in the same group (called a *cluster*) are more similar to each other and less similar to feature vectors of different clusters.

The most obvious measure of similarity between two vectors is the distance between them. If distance is a good measure of dissimilarity, then one would expect distance between feature vectors combined in the same group to be significantly less than distance between vectors in different clusters. The simple Euclidean distance is the most common distance metric used in clustering algorithms. However, clusters defined by Euclidean distance are not invariant to linear transformations in general. The more broad class of metrics is the general *Minkowski metric* of the form

$$d(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{k=1}^d |x_{i_k} - x_{j_k}|^q \right)^{1/q},$$

where $q \geq 1$. Another alternative is to use metric whose associated norm is based on the data itself. For example *Mahalonobis distance*

$$r(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)' \Sigma^{-1} (\mathbf{x}_i - \mathbf{x}_j)},$$

where Σ^{-1} is the inverse of the covariance matrix of the dataset.

In the rest of the section we will discuss three different approaches to the unsupervised learning which cover most of the general approaches used in developing clustering algorithms. First are the *Hierarchical* methods which construct the hierarchy of clusters embedded one into another. Second, is, perhaps, the most broad type of clustering algorithms which are based on function optimization. There objects are grouped in the way which optimize the predefined cost function. The last type of algorithm is based on the analysis of vector connectivity graphs, where nodes are the data points and edges correspond to the similarity relation between two points.

1.4.1 Hierarchical algorithms

Let $D = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ be the set of feature vectors. Consider a sequence of partitions of the set D into n clusters. Initially, there are N clusters each containing just one sample from D . Next, we partition D into $N - 1$ clusters, then into $N - 2$ and so on until all the samples grouped in one cluster. We will say that we are at level k in the sequence of the partitions when number of clusters $n = N - k + 1$.

Definition 1.4.1. *The sequence of partitions of the set D into clusters is called hierarchical if it has the property, that if two samples in the same cluster at the level k they remain in the same cluster at levels $k + i$, $i = 1, \dots, N - k$.*

Often cluster hierarchy is represented by binary trees, where leaves correspond to

PROCEDURE Agglomerative Hierarchical Clustering

```

1:      Initialize number of clusters  $K$ 
2:      Initialize  $\mathcal{D} = \{\{\mathbf{x}_1\}, \dots, \{\mathbf{x}_N\}\}$ 
3:      WHILE  $|\mathcal{D}| > K$  DO
4:          Select  $\mathcal{C}_i, \mathcal{C}_j \in \mathcal{D}$  such that
5:               $d(\mathcal{C}_i, \mathcal{C}_j) = \min\{d(\mathcal{C}_k, \mathcal{C}_l) \mid \mathcal{C}_k, \mathcal{C}_l \in \mathcal{D}, k \neq l\}$ 
6:          Set  $\mathcal{D} = (\mathcal{D} - \{\mathcal{C}_i, \mathcal{C}_j\}) \cup \{\mathcal{C}_i \cup \mathcal{C}_j\}$ 
7:      END WHILE

```

Figure 1.2: Agglomerative hierarchical clustering algorithm combines clusters into larger ones using similarity measure between sets of points.

the N initial clusters and the root is the final cluster containing all the points from the dataset. Each node corresponds to a cluster $\mathcal{C} = \mathcal{C}_l \cup \mathcal{C}_r$, where $\mathcal{C}_l$ and $\mathcal{C}_r$ are the clusters corresponding to the left and right children of the node $\mathcal{C}$. The tree has at most N levels.

There are two types of hierarchical clustering algorithms: *Agglomerative* and *Divisive* algorithms. The difference is in the direction in which the partitioning sequence is constructed. Agglomerative algorithms start with the initial N points and continue by combining points into larger clusters like it was described above. On the contrary, divisive algorithms start with the whole set D and repeatedly bi-partition it until the stopping criterion is met.

The standard agglomerative clustering algorithm is given in the Figure 1.2. We assume that the expected number of clusters $K \geq 1$ is given. The algorithm starts by putting each point into a separate cluster. On each iteration step two nearest among the current clusters combined together. The algorithm stops when the predefined

number of clusters has been obtained.

Obviously cluster structure depends on the distance measure d used by the algorithm. Below we give some common metrics or pseudo-metrics used in clustering to measure the distance between sets of points:

$$\begin{aligned} d_{min}(\mathcal{C}_i, \mathcal{C}_j) &= \min_{\mathbf{x}_i \in \mathcal{C}_i, \mathbf{x}_j \in \mathcal{C}_j} \|\mathbf{x}_i - \mathbf{x}_j\|, \\ d_{max}(\mathcal{C}_i, \mathcal{C}_j) &= \max_{\mathbf{x}_i \in \mathcal{C}_i, \mathbf{x}_j \in \mathcal{C}_j} \|\mathbf{x}_i - \mathbf{x}_j\|, \\ d_{avg}(\mathcal{C}_i, \mathcal{C}_j) &= \frac{1}{|\mathcal{C}_i| \cdot |\mathcal{C}_j|} \sum_{\mathbf{x}_i \in \mathcal{C}_i} \sum_{\mathbf{x}_j \in \mathcal{C}_j} \|\mathbf{x}_i - \mathbf{x}_j\|, \\ d_{mean}(\mathcal{C}_i, \mathcal{C}_j) &= \|\mathbf{m}_i - \mathbf{m}_j\|, \end{aligned}$$

where $\mathbf{m}_c$ is the mean vector of the points in $\mathcal{C}_c$.

All the distance measures defined above tend to minimize the within-cluster variance and produce similar results when clusters are compact and well separated. However, if the clusters are close to one another, or if their shapes are not described by a hypersphere, the different metrics could yield quite different results. The choice of what measure to use is problem specific and should be evaluated experimentally.

1.4.2 Clustering algorithms based on function optimization

One of the main procedures in probabilistic clustering is based on finding parameters of a mixture probability density function which maximizes the likelihood that the observed data sample was drawn from that mixture. We assume that we know the number of clusters K and the form of class-conditional probability densities $p(\mathbf{x} | \mathcal{C}_i, \theta_i)$ with unknown parameters θ_i , $i = 1, \dots, K$. The samples were obtained by, first, selecting the state of nature $\mathcal{C}_i$ with probability $P(\mathcal{C}_i)$ and then selecting $\mathbf{x}$ from the probability density $p(\mathbf{x} | \mathcal{C}_i, \theta_i)$.

The probability density function of the samples is given by

$$p(\mathbf{x} \mid \Theta) = \sum_{i=1}^K p(\mathbf{x} \mid \mathcal{C}_i, \theta_i) P(\theta_i),$$

where $\Theta = (\theta_1, \dots, \theta_K)'$.

The likelihood of the observed dataset $D = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ given parameters Θ

$$p(D \mid \Theta) = \prod_{i=1}^N p(\mathbf{x}_i \mid \Theta).$$

The maximal likelihood estimate $\hat{\Theta}$ is the value of Θ which maximizes $P(D \mid \Theta)$, i.e.

$$p(D \mid \Theta = \hat{\Theta}) = \max\{P(D \mid \Theta = \Theta') \mid \forall \Theta'\}.$$

The log-likelihood

$$L(\Theta) = \sum_{i=1}^N \ln p(\mathbf{x}_i \mid \Theta).$$

If we assume that $p(D \mid \Theta)$ is differentiable function of Θ , we can compute its partial derivative with respect to Θ

$$\frac{\partial L(\Theta)}{\partial \Theta} = \sum_{i=1}^N \frac{\partial \ln p(\mathbf{x}_i \mid \Theta)}{\partial \Theta} = \sum_{i=1}^N \frac{1}{p(\mathbf{x}_i \mid \Theta)} \cdot \frac{\partial p(\mathbf{x}_i \mid \Theta)}{\partial \Theta}$$

which should vanish at $\hat{\Theta}$.

If we assume that parameters θ_k and θ_c , $k \neq c$ are functionally independent and introduce posterior probabilities

$$P(\mathcal{C}_k \mid \mathbf{x}_i, \Theta) = \frac{p(\mathbf{x}_i \mid \mathcal{C}_k, \theta_k) P(\mathcal{C}_k)}{p(\mathbf{x}_i \mid \Theta)}$$

we will get the following expression for the gradient of $L(\Theta)$ with respect to θ_k

$$\begin{aligned}
\frac{\partial L(\Theta)}{\partial \theta_k} &= \sum_{i=1}^N \frac{1}{p(\mathbf{x}_i | \Theta)} \cdot \frac{\partial p(\mathbf{x}_i | \Theta)}{\partial \theta_k} \\
&= \sum_{i=1}^N P(\mathcal{C}_k | \mathbf{x}_i, \Theta) \cdot \frac{1}{p(\mathbf{x}_i | \mathcal{C}_k, \theta_k) P(\mathcal{C}_k)} \cdot \frac{p(\mathbf{x}_i | \Theta)}{\partial \theta_k} \\
&= \sum_{i=1}^N P(\mathcal{C}_k | \mathbf{x}_i, \Theta) \cdot \frac{1}{p(\mathbf{x}_i | \mathcal{C}_k, \theta_k) P(\mathcal{C}_k)} \cdot \frac{\partial}{\partial \theta_k} \sum_{c=1}^K p(\mathbf{x}_i | \mathcal{C}_c, \theta_c) P(\mathcal{C}_c) \\
&= \sum_{i=1}^N P(\mathcal{C}_k | \mathbf{x}_i, \Theta) \cdot \frac{\partial \ln p(\mathbf{x}_i | \mathcal{C}_k, \theta_k)}{\partial \theta_k}.
\end{aligned}$$

The gradient must vanish at the value of θ_k which maximizes $L(\Theta)$ and therefore the maximal likelihood estimate $\hat{\theta}_k$ must satisfy condition

$$\sum_{i=1}^N P(\mathcal{C}_k | \mathbf{x}_i, \hat{\Theta}) \cdot \frac{\partial \ln p(\mathbf{x}_i | \mathcal{C}_k, \hat{\theta}_k)}{\partial \theta_k}, k = 1, \dots, K.$$

Now, let $\hat{P}(\mathcal{C}_k)$ be the maximum-likelihood estimate for the prior probability $P(\mathcal{C}_k)$. It can be shown that

$$\hat{P}(\mathcal{C}_k) = \frac{1}{N} \sum_{i=1}^N \hat{P}(\mathcal{C}_k | \mathbf{x}_i, \hat{\theta}),$$

where

$$\hat{P}(\mathcal{C}_k | \mathbf{x}_i, \theta_k) = \frac{p(\mathbf{x}_i | \mathcal{C}_k, \hat{\theta}_k) \hat{P}(\mathcal{C}_k)}{\sum_{c=1}^K p(\mathbf{x}_i | \mathcal{C}_c, \hat{\theta}_c) \hat{P}(\mathcal{C}_c)}$$

and such that

$$\sum_{i=1}^N \hat{P}(\mathcal{C}_k | \mathbf{x}_i, \hat{\Theta}) \frac{\partial \ln p(\mathbf{x}_i | \mathcal{C}_k, \hat{\theta}_k)}{\partial \theta_k} = 0.$$

Now we can introduce the *Generalized Mixture Decomposition Algorithm* which learns the underlying mixture density by finding the maximum-likelihood estimates $\hat{\theta}_k$ and $\hat{P}(\mathcal{C}_k)$ (see Figure 1.3).

PROCEDURE Generalized Mixture Decomposition

- 1: Choose initial estimates for $\theta_k^0, P^0(\mathcal{C}_k), k = 1, \dots, K$
 - 2: **REPEAT**
 - 3: Expectation step:
 - 4: Set $P(\mathcal{C}_k | \mathbf{x}_i, \Theta^t) = \frac{p(\mathbf{x}_i | \mathcal{C}_k, \theta_k^t) P^t(\mathcal{C}_k)}{\sum_{c=1}^K p(\mathbf{x}_i | \mathcal{C}_c, \theta_c^t) P^t(\mathcal{C}_c)},$
 - 5: $i = 1, \dots, N, k = 1, \dots, K.$
 - 6: Maximization step:
 - 7: Set θ_k^{t+1} to solutions of equation
 - 8: $\sum_{k=1}^K \sum_{i=1}^N P(\mathcal{C}_k | \mathbf{x}_i, \Theta^t) \frac{\partial}{\partial \theta_k} p(\mathbf{x}_i | \mathcal{C}_k, \theta_k^t) = 0$
 - 9: with respect to $\theta_k, k = 1, \dots, K.$
 - 10: Set priors $P^{t+1}(\mathcal{C}_k) = \frac{1}{N} \sum_{i=1}^N P(\mathcal{C}_k | \mathbf{x}_i, \Theta^t), k = 1, \dots, K.$
 - 11: **UNTIL** $\|\Theta^{t+1} - \Theta^t\| < \epsilon$
-

Figure 1.3: Generalized Mixture Decomposition Algorithm.

This algorithm is guaranteed to converge to a minimum (global or local). It is commonly used in the case where the component densities are multivariate normal

$$p(\mathbf{x} \mid \mathcal{C}_k, \theta_k) \sim \mathcal{N}(\mu_k, \Sigma_k),$$

where μ_k, Σ_k are unknown parameters θ_k .

The classification decision regarding vector $\mathbf{x}$ is made according to the final estimates $P(\mathcal{C}_k \mid \mathbf{x})$. This is a typical Bayesian decision rule:

$$\text{Decide } \mathbf{x} \text{ is in } \mathcal{C}_k \text{ if } P(\mathcal{C}_k \mid \mathbf{x}) = \max_{c=1, \dots, K} P(\mathcal{C}_c \mid \mathbf{x}).$$

Given that we know the form of component probability densities, which are differentiable with respect to θ_k the GMD algorithm usually yields satisfactory results. Unfortunately, it is not always possible to predict explicit cluster probability densities. Moreover, solving equations on each iteration of the algorithm could be time consuming. For example in the case of normal densities, we would have to compute and invert sample covariance matrices on each iteration.

There are various iterative procedures used to simplify the computation and speed up the convergence. Such techniques optimize criterion functions which measure the clustering quality of different partitions of the given set D . The problem of finding optimal clustering is reduced to finding the partition which optimizes the criterion function.

The simplest and most widely used criterion function is the one which minimizes the sum-of-squared-error

$$J_e = \sum_{k=1}^K \sum_{\mathbf{x} \in \mathcal{C}_k} \|\mathbf{x} - \mathbf{m}_k\|^2,$$

where $\mathbf{m}_k$ is the sample mean of the points in cluster $\mathcal{C}_k$. This function assumes that for a given cluster $\mathcal{C}_k$ its mean $\mathbf{m}_k$ is the best representative of vectors in $\mathcal{C}_k$. The

PROCEDURE *k*-means Clustering

- 1: Choose initial centers $\mu_1^0, \dots, \mu_K^0$
 - 2: **REPEAT**
 - 3: Set $\mathcal{C}_k^t = \{\mathbf{x} \mid \|\mathbf{x} - \mu_k^t\| \leq \|\mathbf{x} - \mu_m^t\|, \mathbf{x} \in D, m = 1, \dots, K\}$
 - 4: $\mu_k^{t+1} = \frac{1}{|\mathcal{C}_k^t|} \sum_{\mathbf{x} \in \mathcal{C}_k^t} \mathbf{x}$
 - 5: **UNTIL** $J_e^{t+1} = J_e^t$
-

Figure 1.4: *k*-means clustering procedure.

famous *k*-means procedure described below finds clusters by optimizing the criterion function J_e .

The generalization of function J_e is called Related Minimum Variance Criteria

$$J_{min} = \frac{1}{2} \sum_{k=1}^K |\mathcal{C}_k| \min_{\mathbf{x}_i, \mathbf{x}_j \in \mathcal{C}_k} d(\mathbf{x}_i, \mathbf{x}_j),$$

where d is a similarity (distance) measure. Depending on the choice of the distance d clusters obtained by minimizing J_{min} can be significantly different from the ones obtained by minimizing J_e .

One of the most commonly used clustering schemes is called *k*-means. It is an iterative method which minimizes sum-of-squared-error criterion J_e .

It begins with a set of K cluster centers $\mu_1^0, \dots, \mu_K^0$ (see Figure 1.4). Initially these are chosen at random. At iteration t each feature vector is assigned to the cluster This forms the cluster sets $\mathcal{C}_1^t, \dots, \mathcal{C}_K^t$. Then each cluster center is redefined as the mean of the feature vectors assigned to the cluster. Each iteration reduces the criterion function J_e^t :

$$J_e^t = \sum_{k=1}^K \sum_{\mathbf{x} \in \mathcal{C}_k^t} \|\mathbf{x} - \mu_k^t\|^2.$$

As this criterion function is bounded below by zero, the iterations must converge. This method works well when clusters are mutually exclusive and compact around their center means.

1.4.3 Clustering algorithms based on graph theory

Clustering algorithms based on graph theory use a connectivity approach based on the pairwise distance between the samples. Such methods may discover clusters of arbitrary shapes which are not limited to compact blob-like clusters. However, it has been shown that methods based on the shortest distance approach perform poorly when significant amount of noise is present in the data.

Denote by $\Gamma(V, E)$ the complete distance graph of the set D . The set of vertices V corresponds to points from $D = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$. Each edge $e = (\mathbf{x}_i, \mathbf{x}_j)$ is assigned weight $w_e = d(\mathbf{x}_i, \mathbf{x}_j)$.

The similarity graph $\Gamma_S(V, E)$ is obtained from Γ by removing all *inconsistent* edges. By inconsistent we mean edges with "unusually" large weights compared with the others. To be more formal the edge e is inconsistent if $w_e > \theta$, where θ is some threshold value.

Obviously, the similarity relation will depend on the procedure which determines the threshold θ . One approach is to estimate θ by looking at the distribution of the inter-point distances or the weights of the edges of the minimal spanning tree of Γ . If the assumption that the distance measure d used to construct the graph represents similarity between the cluster points is true, we should see a multi-modal distance distribution. Parameter θ can be estimated by taking the distance value, corresponding to the gaps between modes of the distribution.

PROCEDURE MST Clustering

- 1: Construct a complete distance graphs Γ
 - 2: Find a Minimal Spanning Tree $\text{MST}(\Gamma)$
 - 3: Remove inconsistent edges from $\text{MST}(\Gamma)$
 - 4: Identify clusters as connected components of $\text{MST}(\Gamma)$
-

Figure 1.5: Minimal Spanning Tree Clustering algorithm.

Another approach estimates θ for each edge separately by looking at the weight distribution of its neighborhood. A k -neighborhood N_e of the edge e is the set of all edges k steps away from e . We can compute the mean m_e and the standard deviation σ_e of the weights of edges in N_e . The edge e considered to be inconsistent if $w_e > m_e + q\sigma_e$, where q is some number, typically 2.

The simple single-linkage clustering algorithm (see Figure 1.5) reduces the problem to finding the connected components of the graph Γ_S . The two points $\mathbf{x}_i$ and $\mathbf{x}_j$ are considered to be similar to each other, if there is a path in the graph that connects $\mathbf{x}_i$ to $\mathbf{x}_j$. It constructs a minimal spanning tree of Γ and removes the inconsistent edges. Clusters defined as the connected components of the MST.

In the complete-linkage algorithm all samples in the same cluster must be similar to each other. This clustering corresponds to maximally complete subgraphs of the similarity graph Γ_S .

1.5 Stochastic search techniques

1.5.1 The general model

Let $\mathcal{S}$ be the problem search space and $\mathcal{R}$ is a space of possible “cost” values (often $\mathcal{R} = \mathbb{R}$).

Define an objective (“cost”) function f as a map:

$$f : \mathcal{S} \mapsto \mathcal{R}$$

By $\mathcal{F} = \mathcal{R}^{\mathcal{S}}$ we denote the space of all possible problems.

Definition 1.5.1. *The optimization (search) problem:*

$$\text{Find } s \in \mathcal{S}, \text{ s.t. } f(s) = \max(\min)\{f(s_i) | s_i \in \mathcal{S}\}$$

The following is an example of the optimization problem for the famous *travelling salesman problem*:

Problem 1.5.1 (TSP). *Given a finite number of “cities” along with the cost of travel between each pair of them, find the cheapest way of visiting all the cities and returning to your starting point.*

Let $\Gamma = (V, E)$ be a weighted graph with the vertex set corresponding to the “cities” in TSP. Every edge $e = (v_i, v_j) \in E$ has a weight $w(e)$ equal to the cost of travelling from the city v_i to v_j . Now let $\rho = \langle \rho_1, \dots, \rho_N \rangle$ be a permutation of nodes in V . It defines one of the solutions with the cost

$$\text{cost}(\rho) = \sum_{i=1}^{N-1} w((\rho_i, \rho_{i+1})).$$

The search space $\mathcal{S}_\Gamma$ corresponding to the problem is the set of all permutations of nodes in V . The objective function

$$f_\Gamma : \mathcal{S}_\Gamma \mapsto \mathbb{R}$$

defined as the cost of the tour corresponding to a particular permutation from $\mathcal{S}_\Gamma$.

The goal of the search is to

$$\text{Find } s \in \mathcal{S}_\Gamma, \text{ s.t. } f_\Gamma(s) = \min\{f(s_i) \mid s_i \in \mathcal{S}_\Gamma\}.$$

There is a general model that is suitable for almost all heuristic search algorithms [50, 65]. The model gives a nice representation of a search algorithm in terms of the first-order Markov chain.

Define $X \in \mathcal{S}^r$ as a population of possible solutions of size r .

Let

$$\Lambda = \{\mathbf{x} \mid \mathbf{x} \in \mathbb{R}^{|\mathcal{S}|}, x_i \geq 0, \sum_{i \in \mathcal{S}} x_i = 1\},$$

where x_i is the i th component of $\mathbf{x}$. We can define a correspondence between element $\mathbf{x}$ of Λ and a population according to the following rule for defining its components:

$$x_j \text{ is the proportion in the population of the } j\text{th element of } \mathcal{S}.$$

Λ also can be seen as a set of sampling distributions over $\mathcal{S}$, where x_j is the probability of j th element in $\mathcal{S}$ being sampled.

One can easily define a map from the set of all populations into Λ

$$P : \mathcal{S}^r \mapsto \Lambda$$

We also define a heuristic function (the main component of any heuristic algorithm):

$$\mathcal{G} : \Lambda \mapsto \Lambda.$$

Then the model for the (random) heuristic search algorithm (RHS Model) defined as follows.

1. Choose an initial random population X of size r ;
2. Select r independent random samples from the probability distribution $\mathcal{G}(P(X))$, and let these be a new population Y ;
3. Replace X by Y ;
4. Go to step 2.

Now we define the transition probability matrix of the first order Markov Chain corresponding to the RHS model. The states of a Markov Chain correspond to the populations from $\mathcal{S}^r$, there are total of $\binom{|\mathcal{S}| + r - 1}{r}$ such populations.

The transition probability for the Markov chain can be obtained by using the multinomial probability distribution. The sampling space is the set of all ordered sequences from $\mathcal{S}$ of length r . Two sequences are equivalent if one can be obtained from another by reordering of the components. A population corresponds to a class of equivalent sequences.

Let $X_l, X_k \in \mathcal{S}^r$ be populations and let $P(X_l) = p$ and $P(X_k) = q$.

The probability of a sequence from the class corresponding to the population X_k occurring after a sequence from class X_l is

$$\prod_{j \in \mathcal{S}} \mathcal{G}(p)_j^{r q_j}.$$

The total number of sequences from class X_k :

$$r! / \prod_{j \in \mathcal{S}} (r q_j)!$$

Then the transition probability from X_i to X_k is computed from the multinomial distribution

$$P(X_l, X_k) = r! \prod_{j \in \mathcal{S}} \frac{\mathcal{G}(p)_j^{r_{qj}}}{(r_{qj})!}.$$

The RHS model proved to be useful in the analysis of convergence conditions of stochastic search methods. See [18, 56] for more details.

1.5.2 No free lunch theorem

The last two decades has been increasing interest in the application of stochastic search techniques. Many different algorithms and heuristics were developed. The question arises how to compare such algorithms. In particular, is there a class of algorithms that will outperform others? The answer was provided by Wolpert and Macready in the series of so-called “No free lunch theorems” [69, 70].

Consider the first m iterations of a search algorithm represented by a sequence of populations visited during these iterations

$$X_1 \rightarrow X_2 \rightarrow \dots \rightarrow X_m.$$

Let r_m be a vector of corresponding “cost” values

$$r_m = \langle f(X_1), f(X_2), \dots, f(X_m) \rangle, f(X_i) \in \mathcal{R}.$$

We also denote $\Phi(r_m)$ as a performance measure on the set of “cost” values.

The performance of algorithm $\mathcal{A}$ iterated m times on a cost function $f \in \mathcal{F}$ is measured with conditional probability

$$P(r_m \mid f, m, \mathcal{A}).$$

Theorem 1.5.2 (No free lunch theorem for optimization.). *For any pair of algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$,*

$$\sum_f P(r_m \mid f, m, \mathcal{A}_1) = \sum_f P(r_m \mid f, m, \mathcal{A}_2).$$

Corollary 1.5.3. *For any performance measure $\Phi(r_m)$, the average of $P(\Phi(r_m) \mid f, m, \mathcal{A})$ over all f is independent of $\mathcal{A}$.*

The “No free lunch theorem” for optimization claims that if an algorithm does particularly well on one class of problems then it must do more poorly over the remaining problems. In particular, if an algorithm performs better than random search (or total enumeration) on some class of problems then it must perform worse than random search (or total enumeration) on the remaining problems.

The importance of such result is that no general approach can perform better for all possible problems. Another important conclusion is that one cannot develop a good heuristic without intuition and thorough knowledge of the problem.

In the following sections we describe several well known stochastic search methods.

1.5.3 Hill Climbing

Hill climbing is the simplest search algorithm based on the greedy approach. The search is performed by looking at a local “neighborhood” of a particular solution which changes from iteration to iteration.

The simplest implementation of a local search consists of the following steps

1. Pick a candidate solution from the search space. Define it as a current solution.
2. Evaluate all solutions in the neighborhood of the current one. If there is a

solution which is "better" than the current solution, exchange it with the current solution.

3. Repeat step 2 until no solutions in the neighborhood improve the current solution.

The efficiency of the approach depends on the definition of the neighborhood $\mathcal{N} \in \mathcal{S}$. One extreme case is when $\mathcal{N} = \mathcal{S}$. In this case local search performs at least as bad as the standard exhaustive search. If the size of neighborhood is small then we can search it very quickly. However small neighborhoods might result in the algorithm being trapped in local minima. In practice the right thing lies in combining both strategies.

In Figure 1.6 we give a modified version of the standard local search procedure, where the current solution X_c is chosen randomly on each iteration. The best over all solution X_{best} is maintained and compared against newly acquired solutions from the neighborhood of X_c .

The randomness introduced in the algorithm is one of the major tools in fighting the convergence to the local minima (sometimes called premature convergence). Most of the heuristic algorithms, one way or another, make use of random operators to escape the premature convergence.

1.5.4 Simulated Annealing

The simulating annealing algorithm was proposed by Kirkpatrick et al. in 1983 [41]. It introduces a strategy to avoid premature convergence, which is based on the behavior of the annealing process from the statistical mechanics.

PROCEDURE Hill Climbing

```

1:      Initialize  $X_{best} \in \mathcal{S}$ 
2:      REPEAT for a given number of iterations
3:          Select randomly  $X_c \in \mathcal{S}$ 
4:          Select  $X_n \in \mathcal{N}(X_c)$ , s.t.  $eval(X_n) = \max\{eval(X_i) \mid X_i \in \mathcal{N}(X_c)\}$ 
5:          IF  $eval(X_n) > eval(X_c)$ 
6:              SET  $X_{best} = X_n$ 
7:          ELSE
8:              SET  $X_{best} = X_c$ 
9:      END REPEAT

```

Figure 1.6: Hill Climbing.

Annealing is the physical process of heating up solid and the slowly cooling it down until it crystallizes. The atoms have higher energies when the temperature is high and have more freedom to arrange themselves. When the temperature goes down the atomic energy decreases. A crystal with regular structure is obtained when the system reaches the minimal energy state. If the cooling was done too quick, then irregularities and defects are seen in the crystal structure. In such case the system ends in a higher energy state.

At a given state (temperature T) the probability distribution of system energies is determined by the Boltzmann probability:

$$P(E) = e^{\frac{-E}{kT}},$$

where E is the system energy, k is Boltzmann's constant and $P(E)$ is the probability that the system in a state with energy E .

The simulated annealing algorithm (see Figure 1.5.4) is based on the analogy between searching of the global minima in the search problem and achieving the minimal energy state in the simulated annealing. As in the case of Hill Climbing, simulated annealing algorithm consists of a sequence of iterations. On each iteration the current solution is modified. However in the case of simulated annealing, selection of the best current solution is done probabilistically. With probability

$$P(X_n) = e^{\frac{eval(X_n) - eval(X_c)}{T}}$$

the current solution X_c can be substituted by a new solution X_n which is evaluated to be "worse" than X_c . This random ("noise") operator allows algorithm to avoid some of the local minima. The *scheduling parameter* T decreases with each iteration allowing more freedom in making random steps on the early stages of the algorithm.

The efficiency of the simulated annealing algorithm depends on the *cooling schedule* i.e. how fast the parameter T decreases. When T reduced slowly, the algorithm will have more chances to hit the global minimum. This is due to the fact that the larger area of the search space is being searched with the slow cooling schedule. The tradeoff is it requires more computation.

1.5.5 Tabu Search

The tabu search algorithm was developed independently by Glover [26, 28, 29, 30] and Hansen [27] in 1986. Intuitively it can be view as the Hill Climbing with dynamic memory. Moves from one solution to another, performed by the algorithm, are recorded in the so-called *tabu list*. Tabu Search avoids local minima by consulting tabu list each time a new move is performed. If best candidate solution has being

PROCEDURE Simulated Annealing

```

1:      Initialize temperature  $T$ 
2:      Select randomly  $X_c \in \mathcal{S}$ 
3:      REPEAT
4:          REPEAT
5:              Select randomly  $X_n \in N(X_c)$ 
6:              IF  $eval(X_c) < eval(X_n)$ 
7:                  SET  $X_c = X_n$ 
8:              ELSE
9:                  SET  $X_c = X_n$  with probability  $e^{\frac{eval(X_n) - eval(X_c)}{T}}$ 
10:             UNTIL termination condition
11:         Update  $T$ 
12:     UNTIL halting condition

```

Figure 1.7: Simulated Annealing (Kirkpatrick et al. 1983).

PROCEDURE Tabu Search

- 1: Select randomly $X_c \in \mathcal{S}$
 - 2: **SET** $TABULIST = \emptyset$
 - 3: **REPEAT**
 - 4: Select $X_n \in N(X_c) - TABULIST$, s.t.
 - 5: $eval(X_n) = \max\{eval(X_i) \mid X_i \in N(X_c) - TABULIST\}$
 - 6: **SET** $TABULIST = TABULIST \cup \{X_n\}$
 - 7: **SET** $X_c = X_n$
 - 8: **UNTIL** halting condition
-

Figure 1.8: Tabu search (Glover, Hansen 1986).

previously visited and recorded in the list, the next best candidate is considered. The algorithm itself is given in Figure 1.5.5.

1.5.6 Genetic Algorithms

Genetic algorithms are stochastic search algorithms driven by a heuristic, which is represented by an evaluation function, and special random operators: crossover, mutation and selection. The first Genetic Algorithm goes back to J.H.Holland in 1975 [37]. This theory has been further developed and now Genetic Algorithms are widely used to solve search and optimization problems [31, 53, 57, 58]. Terminology as well as the idea itself come from biology. Genetic Algorithms may vary in their implementation. There are many different approaches that are used to solve particular problems.

Let $\mathcal{S}$ be a search space. Each solution is represented by a finite string (*chromosome*). To simplify notations we will not distinguish between search space $\mathcal{S}$ and its representation by strings. Originally, solutions were encoded into strings of bits, but usually it is more convenient and effective to use problem-specific representations.

We are looking for an element in $\mathcal{S}$ which is a solution to a given problem. A tuple $X \in \mathcal{S}^r$ (r is a fixed positive integer) is called a *population* and components of P are called *members* of the population.

The algorithm starts by choosing an initial population $X_0 \in \mathcal{S}^r$. Generally, the first population is generated randomly to provide a higher diversity of possible solutions.

On each *iteration* $i = 1, 2, \dots$ Genetic Algorithm produces a new population X_i by means of random operators. The goal is to produce a population which contains a solution to the problem. One iteration of Genetic Algorithm simulates natural evolution. The "cost" function, called *fitness function*,

$$f : \mathcal{S} \rightarrow \mathbb{R}_+$$

implicitly directs this evolution. Members of the current population X_i with higher fitness value have more impact on generating the next population X_{i+1} . The function f measures on how close is a given member m to a solution. To halt the algorithm one has to provide in advance a *termination condition* and check whether it holds or not on each iteration.

During each iteration of an evolutionary algorithm the population is modified by a number of successive probabilistic transformations:

- Selection:

$$sel : \mathcal{S}^r \mapsto \mathcal{S}^p,$$

$2 \leq p < r$, selects p members of a population for recombination, often its a sampling from the population X with probability

$$Pr(m) = \frac{f(m)}{\sum f(m_i)}, m_i \in X;$$

- Recombination:

$$reco : \mathcal{S}^p \mapsto \mathcal{S}^k,$$

$1 \leq k \leq p$, produces a number of “offsprings” from previously selected population members;

- Mutation:

$$mut : \mathcal{S} \mapsto \mathcal{S},$$

performs a random change in the string.

- Replacement:

$$repl : \mathcal{S}^r \times \mathcal{S}^k \mapsto \mathcal{S}^r,$$

construct a new population from old population and the new offsprings.

The algorithm is presented in Figure 1.5.6.

The choice of random operators and evaluating functions is crucial in designing a reliable and efficient genetic algorithm. This requires the problem specific knowledge and a good deal of intuition.

1.6 Hypothesis Testing

One of the main tasks in experimental research is deciding whether the outcome of an experiment represents essential properties of the population or just a random

PROCEDURE Genetic Algorithm

- 1: Initialize current population $X_0 \subset \mathcal{S}^r$
 - 2: **WHILE NOT** termination condition **DO**
 - 3: **SET** New population $X_{i+1} = \text{repl}(X_i, \text{mut}(\text{reco}(\text{sel}(X_i))))$
 - 4: **END WHILE**
-

Figure 1.9: Genetic Algorithm (Holland, 1975).

observation due to the noise or unrepresentative data sample. Statistical hypothesis testing provides a formal way to decide if the results are significant or accidental.

The very general test procedure can be described as follows. We perform experiment to test some statistical hypothesis, called *null hypothesis*, H_0 against the *alternative* hypothesis H_1 . We consider all possible outcomes of the experiment and divide them into two classes: the acceptance class and rejection class. Partitioning is done in such a way, that probability of observing a result in the rejection class is equal to some small value α , called the *significance* level of the test. If the observed outcome of the experiment falls in the acceptance class, we say that we 'accept' the null hypothesis. If the result lies in the rejection class we 'reject' null hypothesis as unsatisfactory.

Consider an example where the experiment's result is a scalar statistic p_e . We would like to test the null hypothesis that p_e was drawn from some known distribution with probability density function f . We define the significance level α and divide area under the density function into acceptance and rejection classes (see Figure 1.10),

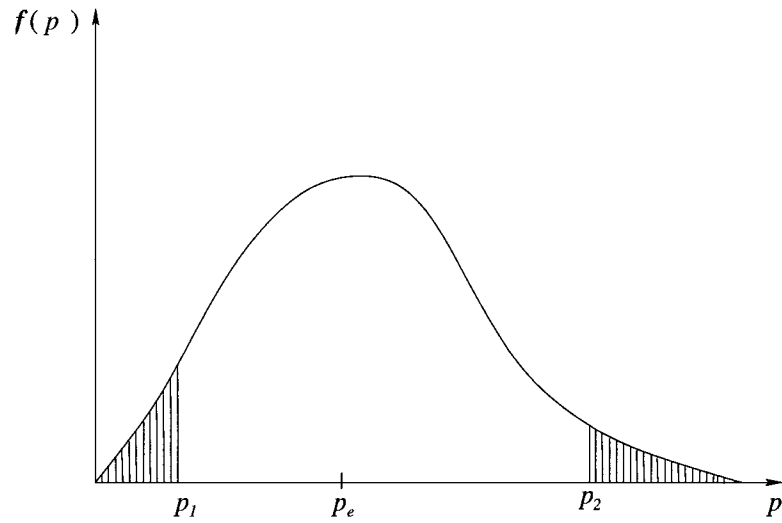


Figure 1.10: Test of hypothesis H_0 .

where

$$\int_{-\inf}^{p_1} f(p)dp = \int_{p_2}^{\inf} f(p)dp = \alpha/2.$$

The shaded area corresponds to the rejection class. In the example we will accept the null hypothesis, since value p_e lies between p_1 and p_2 in the acceptance class.

Below we describe several commonly used tests, designed to test the significance of specific hypothesis.

1.6.1 Pearson's χ^2 test

One of the most commonly used tests to test the agreement between observations and the hypothesis is the so-called Pearson's χ^2 test [51]. It is applied to discrete problems where observations can be grouped into, say, k mutually exclusive classes.

Suppose we made n independent, identically distributed observations. Let n_i and E_i denote the observed and the expected number of observations in the i th class

respectively. The χ^2 statistic defined as

$$\chi^2 = \sum_{i=1}^k \frac{(n_i - E_i)^2}{E_i}. \quad (1.6.1)$$

When there is a perfect agreement, then $n_i = E_i$ and $\chi^2 = 0$. The more the observed and the expected values disagree, the greater the value of the χ^2 statistic.

It can be shown that the χ^2 criterion follows approximately the χ^2 distribution with $k - 1$ degrees of freedom under the assumption that the observed values agree with the hypothesized. Let X_{k-1} be chi-squared distributed with $k - 1$ degrees of freedom. We can compute the probability

$$Prob(\chi^2 > X_{k-1})$$

of the χ^2 statistic be greater than X . We reject the hypothesis if

$$Prob(\chi^2 > X_{k-1}) > 1 - \alpha,$$

where α is the significance level of the test.

1.6.2 k -class mean comparison

Scalar observations

Suppose we have K data samples (groups) of independent scalar observations

$$Y_i = \langle y_1^i, \dots, y_{n_i}^i \rangle, \quad i = 1, \dots, K.$$

We would like to check whether the means of the populations from which these sets were sampled are the same, i.e. we test the null hypothesis H_0 that the true means μ_i are equal, i.e.

$$H_0 : \mu_1 = \mu_2 = \dots = \mu_K$$

against the alternative

$$H_1 : \exists i, j \text{ s.t. } \mu_i \neq \mu_j, \quad i, j = 1, \dots, K.$$

Let

$$\bar{Y}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} y_j^i$$

and

$$\bar{Y} = \frac{1}{N} \sum_{i=1}^K \sum_{j=1}^{n_i} y_j^i,$$

where $N = \sum_{i=1}^K n_i$, i.e. $\bar{Y}_i$ is the sample mean of measurements in Y_i and $\bar{Y}$ is the global mean.

We define *within-group* variance

$$\frac{\sum_{i=1}^K \sum_{j=1}^{n_i} (y_j^i - \bar{Y}_i)^2}{N - K}$$

and *between-group* variance

$$\frac{\sum_{i=1}^K n_i (\bar{Y}_i - \bar{Y})^2}{K - 1},$$

where by "group" we mean a set of measurements Y_i . The ratio of the between-group variance to the within-group variance

$$W = \frac{\sum_{i=1}^K n_i (\bar{Y}_i - \bar{Y})^2 / (K - 1)}{\sum_{i=1}^K \sum_{j=1}^{n_i} (y_j^i - \bar{Y}_i)^2 / (N - K)} \quad (1.6.2)$$

is a well known classic parametric statistic for testing the hypothesis that the means of K normal distributions are the same [59]. If variables $\mathbf{y}_i$ are normally distributed with a common variance, then under the hypothesis H_0 , W has the F -distribution with $K - 1$, $N - K$ degrees of freedom. Unfortunately, statistic W is very sensitive to the normality assumption. Often we don't know the underlying distribution and therefore we cannot assume a normal distribution.

We describe a permutation test based on the statistic W [32]. The assumption we make is that the y_i are independent and distributed with a common, but unknown distribution. Create an ordered set

$$Z = \{y_1^1, \dots, y_{n_1}^1, y_1^2, \dots, y_{n_2}^2, \dots, y_1^K, \dots, y_{n_K}^K\},$$

where $y_k^i \in Y_i$.

Let $Z^p = \langle z_1, \dots, z_N \rangle$ be a permutation of elements in Z . Partition set Z^p into subsets

$$Z_1 = \{z_1, \dots, z_{n_1}\}, Z_2 = \{z_{n_1+1}, \dots, z_{n_1+n_2}\}, \dots, Z_P = \{z_{n_{K-1}+1}, \dots, z_{n_{K-1}+n_K}\}$$

and compute statistic

$$W_p = \frac{\sum_{i=1}^K n_i (\bar{Z}_i - \bar{Z})^2 / (K - 1)}{\sum_{i=1}^K \sum_{j=1}^{n_i} (z_j^i - \bar{Z}_i)^2 / (N - K)},$$

where $z_j^i \in Z_i$ and $\bar{Z}_i, \bar{Z}$ are the group and total means, respectively.

By computing values W_p for all possible permutations of Z we obtain the exact p-value p_0 of the test

$$p_0 = \frac{|\{W_p | W_p > W\}|}{M},$$

where M is the total number of permutations in Z .

We reject the hypothesis if

$$Prob(W \geq W_p | H_0) < \alpha,$$

or, equivalently, when

$$p_0 < \alpha,$$

where α is the significance level of the test.

The number of permutations grows exponentially with the size of Z . In practice, it is almost never possible to compute the exact p-value of the permutation test. We can use a Monte Carlo or Bootstrap approach in order to reduce the amount of computations. We compute the estimated p-value $\hat{p}$ of the test from M randomly chosen permutations of Z . The important question that has to be answered: how can we estimate the number M of sampled permutations.

Let W_{p_i} be the statistic computed from the i th permutation of the set Z . Denote

$$\delta_i = \begin{cases} 1, & W_{p_i} > W, \\ 0, & \text{otherwise.} \end{cases}$$

The estimate for the p-value can be rewritten

$$\hat{p} = \frac{1}{M} \sum_{i=1}^M \delta_i.$$

Therefore, to estimate the appropriate number of permutations to be sampled, we need to estimate the size M of the boolean set $\langle \delta_1, \dots, \delta_M \rangle$. One of the solutions to this problem is given in Section 3.3.1.

Multivariate observations

The test for the equivalence of mean vectors of K multivariate samples can be obtained in the similar way.

Let

$$\mathbf{Y}_i = \langle \mathbf{y}_1^i, \dots, \mathbf{y}_{n_i}^i \rangle,$$

be the i th group of vector observations $\mathbf{y}_j^i \in \mathbb{R}^M$. We test the hypothesis that the true mean vectors are equal

$$H_0 : \mathbf{m}_1 = \mathbf{m}_2 = \dots = \mathbf{m}_K.$$

The alternative hypothesis H_1 will be that at least two of the group means are different.

Again, we define group and global means

$$\bar{\mathbf{Y}}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} \mathbf{y}_j^i$$

and

$$\bar{\mathbf{Y}} = \frac{1}{N} \sum_{i=1}^K \sum_{j=1}^{n_i} \mathbf{y}_j^i$$

respectively. Define the within-group scatter matrix

$$\mathbf{E} = \sum_{i=1}^K \sum_{j=1}^{n_i} (\mathbf{y}_j^i - \bar{\mathbf{Y}}_i)(\mathbf{y}_j^i - \bar{\mathbf{Y}}_i)'$$

and the between group scatter matrix

$$\mathbf{H} = \sum_{i=1}^K n_i (\bar{\mathbf{Y}}_i - \bar{\mathbf{Y}})(\bar{\mathbf{Y}}_i - \bar{\mathbf{Y}})'$$

The within-group and between-group scatter matrices are sometimes called the hypothesized square error and the total square error respectively.

One can explore the ratio $(\mathbf{H} \cdot \mathbf{E}^{-1})$ of the hypothesized error and the total error in order to make judgments about the hypothesis H_0 . The problem, of course, is that $(\mathbf{H} \cdot \mathbf{E}^{-1})$ is a matrix and we need a scalar statistic to be able to perform a permutation test. Notice that eigenvalues of $(\mathbf{H} \cdot \mathbf{E}^{-1})$ represent the ratios of sums of squares corresponding to the canonical variates. Let $\langle \lambda_1, \dots, \lambda_K \rangle$ be the vector of eigenvalues of $(\mathbf{H} \cdot \mathbf{E}^{-1})$ and $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_K$. There were suggested several statistics based on the eigenvalues λ_i . The most commonly used are the following:

1. Pillai-Barlett trace: $\sum_{i=1}^K \lambda_i / (1 + \lambda_i)$;
2. Wilks's lambda: $\prod_{i=1}^K (1 + \lambda_i)^{-1}$;

3. Hotelling-Lawley trace: $\sum_{i=1}^K \lambda_i$;

4. Ray's largest eigenvalue: λ_1 .

In the case of the $K = 1$ all the statistics above become equivalent. The permutation test based on statistics 1-4 can be performed in the same way as was described in the scalar case.

1.6.3 Combining statistics for multivariate observations

Let $\mathbf{Y} = \langle \mathbf{y}_1, \dots, \mathbf{y}_N \rangle$ be a set of multivariate observations on M variables. Now, let $T = \langle T_1, \dots, T_M \rangle$ be the set of single-variable test statistics computed on each component of vectors $\mathbf{y}_i$ separately and $P = \langle p_1, \dots, p_M \rangle$ be the p -values of the corresponding tests. Statistics T might include any statistic one might employ when testing just one variable at a time. Notice we do not require the domains of components in $\mathbf{y}$ to be the same. In fact some of them could be, for example, categorical and some numerical.

The simplest and the most conservative way to combine statistics T in one test is to use Bonferroni correction [7] where the hypothesis is rejected if

$$\min\{p_i | p_i \in P\} < \alpha/M. \quad (1.6.3)$$

Since the approach is very simple, it is widely used in practice. Indeed, in many applications we have to reject the hypothesis if at least one of the observed variables disagrees with the hypothesized values.

Another generalized approach for permutation tests based on the Fisher's omnibus method was introduced by Pesarin [52]. We generate S permutations of $\mathbf{Y}$ and thus obtain S vectors of test statistics denoted by T'_i , $i = 1, \dots, S$. By T'_{ij} we denote the

j th components of the vector T'_i . To obtain a composite multivariate test we need to convert these separate and quite different statistics to a common basis. This is done by ranking the single variable statistics:

$$R_{ij} = R(T'_{ij}) = \sum_{k=1}^S I(T'_{kj} \leq T'_{ij}), \quad i = 1, \dots, S; \quad j = 1, \dots, M,$$

where $I(F)$ is an indicator function, taking values 1 or 0 depending on whether F is true or not.

Next step is to combine the ranks of the M individual tests using Fisher's omnibus method

$$U_i = - \sum_{j=1}^M \log \left(\frac{S + 0.5 - R_{ij}}{S + 1} \right), \quad i = 1, \dots, S.$$

We determine the marginal significance levels of each of the single-variable test:

$$p_j = \frac{0.5 + \sum_{k=1}^S I(T'_{kj} \geq T_j)}{S + 1}, \quad j = 1, \dots, M$$

and combine these values

$$U = - \sum_{i=1}^M \log(p_i).$$

The significance level of the combined test determined as follows:

$$p = \frac{0.5 + \sum_{i=1}^S I(U_i \geq U)}{S + 1}.$$

1.7 System evaluation

1.7.1 Cross-validation and bootstrapping

Whenever we evaluate performance of a classification system, the main question is: "Will the system perform as required on an arbitrary input from the population". If

the system is evaluated on the same dataset as it was trained on, we cannot guarantee that the learned parameters correspond to the properties of the whole population and not just the particular training set. To get reliable and unbiased estimates of the system performance other sets of data that are independent from the training sets must be used. Such sets are called *test datasets*.

If there are no significant limitations on the size of the available data, a simple *split-dataset* technique can be used. We acquire a large set of data and uniformly randomly divide it in two parts. The first part becomes the training set on which the parameters of the system will be estimated and the second becomes the test set for the evaluation.

In the case when large samples of data are not available *cross-validation* and *bootstrapping* techniques are used. Both methods based on "resampling".

In k -fold cross-validation [54, 67], the data is divided into k subsets of (approximately) equal size. The classifier is trained k times, each time leaving out one of the subsets from training. The omitted subset is used as a test set to compute the classification error. If k equals the size of the sample data, this is called *leave-one-out* cross-validation. *Leave-v-out* is a more general and expensive version of cross-validation that involves leaving out all possible subsets of v cases.

Notice that cross-validation is different from the split-datasets case described in the beginning. In the split-sample method, only a single subset (the test dataset) is used to evaluate the system, instead of k different subsets; i.e., there is no "crossing". The distinction between cross-validation and split-sample evaluation is extremely important because cross-validation is markedly superior for small size data sets [34].

Bootstrapping seems to work better than cross-validation in many cases [19, 21,

22], even though it is known that bootstrapping does not work well for some methodologies such as empirical decision trees [13], for which it can be excessively optimistic.

In the simplest form of bootstrapping randomly sampled datasets are repeatedly analyzed. Each subsample is a random sample with replacement from the full sample dataset. The number of samples depends on the problem specifics. Usually from 100 to 200 subsamples give good estimates. For estimating generalization error in classification problems, the .632 bootstrap is one of the currently favored methods [23]. In .632 bootstrap the weighted contribution of the training set error is introduced to correct the bias in classification error of the test set.

1.7.2 Accuracy counts

Accuracy is the simplest and one of the most widely used measures of the classifier performance. It is possible to compute accuracy if a test set with the true class labels is available. Let $D = \langle \mathbf{x}_1, \dots, \mathbf{x}_N \rangle$ be a test data set and $y_1, \dots, y_N$ be the corresponding true class labels. Define

$$K = |\{\mathbf{x}_i \mid \text{decide}(\mathbf{x}_i) = y_i, \mathbf{x}_i \in D\}|$$

as the number of correctly classified elements in D , where $\text{decide}(\mathbf{x})$ is the label assigned by the given classification algorithm. To evaluate the performance of the system we compute the accuracy measure:

$$A = \frac{K}{|D|},$$

which gives the fraction of the correctly classified elements from the test set D .

Notice, that the numbers of correctly classified elements follow the Binomial distribution and A can be viewed as an estimate of probability p of a word being classified

correctly.

We are interested in constructing a confidence interval for probability p . For binomial variates, exact confidence intervals do not exist in general. One can obtain an approximate $100(1 - \alpha)\%$ confidence interval $[p_S, p_L]$ by approximately solving the following equations for p_S and p_L :

$$\sum_{i=0}^K \binom{|D|}{i} p_S^i (1 - p_S)^{|D|-i} = \alpha/2,$$

$$\sum_{i=K}^{|D|} \binom{|D|}{i} p_L^i (1 - p_L)^{|D|-i} = \alpha/2$$

for a given α .

1.7.3 Receiver Operating Characteristic

The *Receiver Operating Characteristic* (ROC) curves were developed in the 1950's and have found great use in experimental psychology, radar detection, medicine and other fields which require decision making in noisy environments [62]. ROC curves indicate the intrinsic properties of a systems' diagnostic performance and can be used to compare the relative performance of competing procedures.

Suppose that the classification decision of the classifier is based on a real valued estimator $\hat{y}$ computed by the system. The classifier employs a threshold θ to determine the class:

if $\hat{y}(\mathbf{x}) \leq \theta$ decide $\mathbf{x}$ in class $\mathcal{C}_1$, otherwise in class $\mathcal{C}_2$.

To follow the commonly used signal detection terminology we will say, that the signal is present if the observation $\mathbf{x}$ is in the class $\mathcal{C}_2$ and not present if $\mathbf{x} \in \mathcal{C}_1$.

The power of such inference could be evaluated by looking at the ratio of the two types of errors:

- *false alarm* - system decided that the signal is present when it was not;
- *misdetect* - system detected no signal when it was present.

By repeating experiments a large number of times we can estimate, for a given threshold value θ , the probabilities of

$$FA = P(\hat{y}(\mathbf{x}) > \theta \mid \mathbf{x} \in \mathcal{C}_1) \text{ and } MD = P(\hat{y}(\mathbf{x}) \leq \theta \mid \mathbf{x} \in \mathcal{C}_2)$$

of the system having false alarm and misdetect errors respectively.

The receiver operating characteristic curve is constructed by varying θ in the range of all possible values and plotting the rates of false alarm errors against the rates of misdetect errors for each θ . An example of ROC curve is given in Figure 1.11. It is easy to see that the smaller the area under the curve the more accurate and informative our decision procedure is.

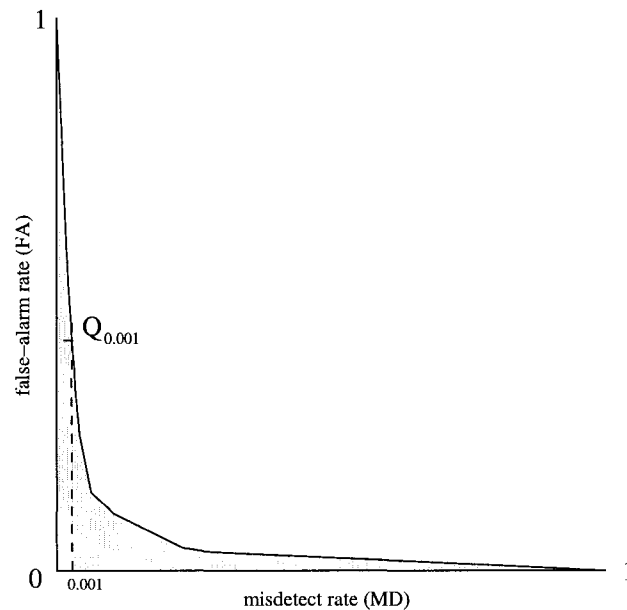


Figure 1.11: Receiver Operating Characteristic.

ROC curves are also very useful, when we want to evaluate the performance of a system with a given rate of false alarm or misdetect. Suppose we are more concerned about the misdetect type of errors. An example of such application could be a missile attack detection, where the cost of missing an attack is much higher than the cost of triggering false alarm. In such case we would like to limit the value of MD (say 0.001) and choose threshold θ , such that

$$P(\hat{y}(\mathbf{x}) \leq \theta \mid \mathbf{x} \in \mathcal{C}_2) \leq 0.001 \text{ and } P(\hat{y}(\mathbf{x}) > \theta \mid \mathbf{x} \in \mathcal{C}_1) \text{ is minimal.}$$

1.7.4 Evaluation of Clustering algorithms

The issue of cluster validity is a complex one [61] further complicated when dealing with real datasets with known class labels, but unknown structure.

An ideal clustering algorithm classifies points such that points assigned the same class label belong to the same cluster and the number of cluster labels and class labels are identical to the ground-truth classification. To evaluate a clustering algorithm, we face the following problem: How can we map the labels assigned by a clustering algorithm to the ground-truth cluster labels in order to make valid comparisons? In addition, once we have established this mapping, is there a measure that will allow us to compare algorithms?

Given the ground-truth labels and the labels determined by some clustering algorithm we form a contingency table by counting the number of times the algorithm assigns a label j to a cluster point when the actual label is i . We form all possible mappings from the smaller of the actual and assigned label sets onto the larger of these two sets and build a confusion matrix for each mapping. Since the diagonal elements of the confusion matrix represent the number of times that an instance of

class i was correctly identified as belonging to class i , we can therefore derive a criterion function that returns a value for each mapping based on the number of correct classifications. The optimal mapping is the one associated with the confusion matrix whose sum along the diagonal (the trace) is maximal.

More formally, given a ground truth test dataset consisting of a collection of N feature vectors and corresponding actual cluster label for each vector, we wish to evaluate the performance of a given clustering algorithm.

Let $D = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, $\mathbf{x} \in \mathbb{R}^d$ be the set of N feature vectors and let $Y = \{y_1, \dots, y_N\}$ such that $y_i \in C$ be the set of corresponding cluster labels, where C is the ordered set of cluster labels or their names.

Let L be the ordered set of possible labels assigned by the clustering algorithm. Note that $|L|$ is not necessarily equal to the $|C|$ and the labels need not have the same representation, for example actual labels may be alphanumeric characters or strings, while assigned labels may be some subset of the integers. Let $A = \{a_1, \dots, a_N\}$, such that $a_i \in L$ for $i = 1, 2, \dots, N$ be the assigned labels that are the output of a clustering algorithm. We form a $|C| \times |L|$ matrix $\mathbf{Z}$, whose (i, j) entry is equal to the number of times that a vector with true label $c_i \in C$ was assigned label $l_j \in L$ by the algorithm, where i and j are the indexes of labels c_i and l_j in the sets C and L respectively.

Given $\mathbf{Z}$, we form all possible permutations of the rows or columns of $\mathbf{Z}$, depending on which is smaller. Let $K = \min(|C|, |L|)$. For the j th permutation we compute

$$v_j = \sum_{i=1}^K \mathbf{Z}^j[i, i],$$

and choose the permutation which gives the maximal sum v_p .

Having found the optimal permutation, we can form the $K \times K$ confusion matrix

$$\mathbf{M} = \mathbf{Z}^p,$$

where p is such that

$$v_p = \max\{v_j \mid \forall j\}.$$

The diagonal elements of $\mathbf{M}$ indicate the number of correct classifications and

$$\frac{1}{N} \sum_{i=1}^K \mathbf{M}_{[i,i]}$$

gives the percentage of points correctly classified, providing a measure of the accuracy of the algorithm.

Chapter 2

The Automorphism Problem in Free Groups

2.1 Introduction

In this chapter we introduce the group theoretic notions and results used throughout this writing. In particular, the Whitehead Reduction Problem, which will be the main subject of discussions in the following chapters is introduced here. In the next section we describe an algorithm to solve the Automorphism problem which follows directly from the results of Whitehead. In Section 2.3 we give complexity results on the general Automorphism and the Whitehead Reduction problems. We also introduce the Length Reduction Problem which seems to be the main obstacle in solving Whitehead Reduction Problem efficiently. At the end of the chapter we suggest a new input size function, called the Whitehead Complexity, which we think is more suitable for analysis of Whitehead-type algorithms.

We begin with a general definition of a group.

Definition 2.1.1. *A group $\langle G, * \rangle$ is a set G with a binary operation $*$, such that the following axioms are satisfied:*

1. Operation $*$ is associative: $a * (b * c) = (a * b) * c$;
2. There is an identity element $e \in G$ such that $\forall a \in G, a * e = e * a = a$;
3. For each $a \in G$ there is an element $a' \in G$ such that $a' * a = a * a' = e$. The element a' is uniquely defined by a . It is called the inverse of a with respect to $*$ and denoted by a^{-1} .

Let $X = \{x_1, \dots, x_n\}$ be a finite set and let $X^\pm = \{x^{\pm 1} | x \in X\}$. A word $\bar{u} = u_1 \dots u_l, u_i \in X^\pm$, is *reduced* if it does not contain a subword xx^{-1} or $x^{-1}x, x \in X$. An *elementary reduction* of a word w consists of deleting a subword $xx^{-1}, x \in X^\pm$ from w . It can be shown that the reduced word $\bar{w}$ can be obtained from w by a sequence of elementary reductions. The word $\bar{w}$ is the unique reduced form of w and does not depend on a particular sequence of elementary reductions. We denote the empty word by e . The empty word is reduced.

Define a multiplication $u * v$ for $u, v \in F(X)$ by $u * v = \overline{uv}$. Now, let $F(X)$ be the set of all reduced words in $X^\pm$ together with an empty word e .

Lemma 2.1.1. *Set $F(X)$ forms a group with respect to multiplication $*$.*

Proof. First we show that operation $*$ is associative, i.e.

$$(u * v) * w = u * (v * w)$$

for any $u, v, w \in F(X)$. We need to prove that

$$\overline{\overline{u * v} * w} = \overline{u * \overline{v * w}}$$

for given u, v, w . Observe, that each of the reduced words $\overline{\overline{u * v} * w}, \overline{u * \overline{v * w}}$ can be obtained from uvw by a sequence of elementary reductions. We have already

mentioned that the reduced form a word is unique and does not depend on the order of reductions, therefore

$$\overline{u * v * w} = \overline{uvw} = \overline{u * \overline{v * w}}.$$

Clearly the empty word e , which is in $F(X)$, is the identity element with respect to $*$

$$e * w = w * e = w$$

for every $w \in F(X)$.

Now let $w = x_1 \dots x_l$, $x_i \in X^{\pm 1}$. Then the word

$$w^{-1} = x_l^{-1} \dots x_1^{-1}$$

is also reduced, therefore it is in $F(X)$ and

$$w * w^{-1} = \overline{x_1 \dots x_l x_l^{-1} \dots x_1^{-1}} = e.$$

Hence, w^{-1} is the inverse of w . This shows that $\langle F(X), * \rangle$ satisfies all the axioms of a group. ■

Definition 2.1.2. A group $\langle F(X), * \rangle$ is called a free group.

The set X is called a *basis* of the free group $F(X)$ and $n = |X|$ is the *rank* of $F(X)$. By F_n we denote a free group of rank n .

For a word u by $|u|$ we denote the length of u , similarly, for a tuple $U = (u_1, \dots, u_k) \in F^k$, we denote by $|U|$ the *total length* $|U| = |u_1| + \dots + |u_k|$, where F^k is the cartesian product of F with itself k times.

A word $w = x_1 \dots x_n \in F(X)$ ($x_i \in X^{\pm 1}$) is *cyclically reduced* if $x_1 \neq x_n^{-1}$. Clearly, every $w \in F$ can be presented in the form $w = u^{-1} \tilde{w} u$ for some $u \in F(X)$ and a cyclically reduced element $\tilde{w} \in F(X)$ such that $|w| = |\tilde{w}| + 2|u|$. This $\tilde{w}$ is unique and it is called a *cyclically reduced form* of w .

Definition 2.1.3. A bijection $\phi : F \rightarrow F$ is called an automorphism of F if $\phi(uv) = \phi(u)\phi(v)$ for every $u, v \in F$.

The set $Aut(F)$ of all automorphisms of F forms a group with respect to composition of automorphisms. Every automorphism $\phi \in Aut(F)$ is completely determined by its images on elements from the basis X since $\phi(x_1 \dots x_n) = \phi(x_1) \dots \phi(x_n)$ and $\phi(x^{-1}) = \phi(x)^{-1}$ for any letters $x_i, x \in X^{\pm 1}$, $i = 1, \dots, n$. For simplicity, we might write $w\phi$ to denote the action of an automorphism ϕ on $w \in F$.

For an automorphism φ of F , and k -tuples $U = (u_1, \dots, u_k), V = (v_1, \dots, v_k) \in F^k$ we write $U\varphi = V$ if $u_i\varphi = v_i$, $i = 1, \dots, k$.

In 1936 J.H.C. Whitehead introduced the following algorithmic problem, which became a central problem of the theory of automorphisms of free groups [68].

Problem 2.1.2 (Whitehead). Given two tuples $U, V \in F^k$ determine if there is an automorphism $\varphi \in Aut(F)$ such that $U\varphi = V$.

In his paper, Whitehead showed (using a topological argument) that this problem can be solved algorithmically and suggested an algorithm to find such an automorphism φ (if it exists). To explain this method we need the following definition.

Definition 2.1.4. An automorphism $t \in Aut(F)$ is called a Whitehead automorphism if it has one of the following types:

- 1) t permutes elements in $X^{\pm 1}$;
- 2) t takes each element $x \in X^{\pm 1}$ to one of the elements $x, xa, a^{-1}x$, or $a^{-1}xa$, where $x \neq a^{\pm 1}$ and $a \in X^{\pm 1}$ is a fixed element.

Denote by $\Omega_n = \Omega(F)$ the set of all Whitehead automorphisms of a given free group $F = F_n(X)$. Nielsen [49] showed that the following set of automorphisms

$$N_n = \{x_i \rightarrow x_i x_j, x_i \rightarrow x_j^{-1} x_i, x_i \rightarrow x_i^{-1} \mid x_i, x_j \in X, i \neq j\} \quad (2.1.1)$$

generates the whole group of automorphisms $\text{Aut}(F_n(X))$. Notice that $N \subset \Omega_n$ and, therefore, Ω_n also generates $\text{Aut}(F_n(X))$.

Let T be a subset of $\text{Aut}(F)$. We say that tuples $U, V \in F^k$ are T -equivalent ($U \sim_T V$) if there exists a finite sequence $t_1, \dots, t_m$ (where $t_i \in T^{\pm 1}$) such that $U t_1 \dots t_m = V$. The T -equivalence class of a tuple U is called the T -orbit $\text{Orb}_T(U)$ of U . If T generates $\text{Aut}(F_n)$ then the equivalence class of a tuple U is called the orbit $\text{Orb}(U)$ of U . Now Problem 2.1.2 can be stated as a membership problem for a given orbit $\text{Orb}(U)$. By $U_{\min}$ we denote any tuple of minimal total length in the orbit $\text{Orb}(U)$, and by $\text{Orb}_{\min}(U)$ - the set of all minimal tuples $U_{\min}$.

It is convenient sometimes to look at Whitehead problem from graph-theoretic view-point.

Definition 2.1.5. *The Automorphism Graph on F^k relative to T is a labelled graph $\Gamma(F, k, T)$ such that:*

1. F^k is the vertex set of Γ ;
2. two vertices $U, V \in F^k$ are connected by a directed edge from U to V with label $t \in T$ if and only if $Ut = V$.

In particular, $\Gamma_k(F) = \Gamma(F, k, \Omega)$ is called *Whitehead's Automorphism Graph* of F . In the case when $k = 1$ we write $\Gamma(F)$ instead of $\Gamma_1(F)$. Obviously, $V \in \text{Orb}(U)$ if and only if U and V are in the same connected component of $\Gamma_k(F)$.

The following theorem is one of the fundamental results in combinatorial group theory.

Theorem 2.1.3 (Whitehead [68]). *Let $U, V \in F_n(X)^k$ and $V \in \text{Orb}(U)$. Then:*

1) *if $|U| > |V|$, then there exists $t \in \Omega_n$ such that*

$$|U| > |Ut|;$$

2) *if $|U| = |V|$, then there exist $t_1, \dots, t_m \in \Omega_n$ such that*

$$Ut_1 \dots t_m = V$$

and $|U| = |Ut_1| = |Ut_1 t_2| = \dots = |Ut_1 t_2 \dots t_m| = |V|$.

In view of Theorem 2.1.3 Problem 2.1.2 can be divided into two subproblems:

Problem 2.1.4 (Reduction Problem RP). *For a tuple $U \in F^k$ find a sequence $t_1, \dots, t_m \in \Omega_n$ such that $Ut_1 \dots t_m = U_{\min}$.*

Problem 2.1.4 is also known as Whitehead descent or Whitehead minimization problem.

Problem 2.1.5 (Minimal Orbit Search Problem MSP). *For tuples $U, V \in F^k$ with*

$$|U| = |U_{\min}| = |V_{\min}| = |V|$$

find a sequence $t_1, \dots, t_m \in \Omega_n$ such that $Ut_1 \dots t_m = V$.

Theorem 2.1.3 gives a solution to the both problems above, and hence to Problem 2.1.2.

Whitehead minimization problem has another important application in deciding whether an element belongs to some generating basis X' of a free group $F(X)$.

Definition 2.1.6. *Elements from the orbit $\text{Orb}(x_i)$, where $x_i \in X$, are called primitive in $F(X)$*

It is known that element v is part of a generating basis if and only if it is a primitive element. The solution to the Whitehead minimization problem allows one to effectively decide whether an element v is primitive or not which is equivalent to showing that $v_{\min} \in X^{\pm 1}$.

2.2 Whitehead Algorithm

The procedures described below give algorithmic solutions to the Problems 2.1.4 and 2.1.5, together they are known as *Whitehead Algorithm* or *Whitehead Method*.

2.2.1 Decision algorithm for RP

Following Whitehead we describe below a deterministic decision algorithm for Problem 2.1.4, we refer to this algorithm (and to its various modifications) as to DWR (“D” stands for deterministic). This algorithm executes consequently the following

Procedure 2.2.1 (Elementary Length Reduction ELR). *Let $U \in F^k$. ELR finds some $t \in \Omega_n$ with $|Ut| < |U|$ (if it exists). Namely, ELR performs the following search. For each $t \in \Omega_n$ compute the length of the tuple Ut until $|U| > |Ut|$, then output Ut . Otherwise stop and output $U_{\min} = U$.*

DWR performs ELR on U , then performs ELR on Ut , and so on, until a minimal tuple $U_{\min}$ is found. We refer to algorithms of this type as to *Whitehead’s descent method* with respect to the set Ω_n .

Clearly, there could be at most $|U|$ repetitions of ELR:

$$|U| > |Ut_1| > \dots > |Ut_1 \dots t_l| = U_{min}, \quad l \leq |U|.$$

The sequence $t_1, \dots, t_l$ is a solution to Problem 2.1.4.

2.2.2 Decision algorithm for MSP.

Here we describe a deterministic decision algorithm for Problem 2.1.5, which is also due to Whitehead. Below we refer to this algorithm as to DWMS.

Let $U, V \in F^k$. DWMS constructs $Orb_{min}(U)$ (as well as $Orb_{min}(V)$) by repeating consequently the following

Procedure 2.2.2 (Local Search LS). *Let $\Omega_n = \{t_1, \dots, t_m\}$ and Δ be a finite graph with vertices from F^k . Given a vertex W in Δ the local search at W results in a graph Δ_W which contains Δ . We define Δ_W recursively. Put $\Gamma_0 = \Delta$, and suppose that Γ_i has been already constructed. If $|Wt_{i+1}| = |W|$ and Wt_{i+1} does not appear in Γ_i then add Wt_{i+1} as a new vertex to Γ_i , also add a new edge from W to Wt_{i+1} with label t_{i+1} , and denote the resulting graph by Γ_{i+1} . Otherwise, put $\Gamma_{i+1} = \Gamma_i$. The routine stops in m steps and results in a graph Γ_m . Put $\Delta_W = \Gamma_m$.*

The construction of $Orb_{min}(U)$ is a variation of the standard Breadth-First search method.

Procedure 2.2.3 (Breadth-First Search BFS). *Start with a graph Δ_0 consisting of a single vertex U . Put $\Delta_1 = (\Delta_0)_U$ and "mark" the vertex U . If a graph Δ_i has been constructed, then take any unmarked vertex W in Δ_i within the shortest distance from U , put $\Delta_{i+1} = (\Delta_i)_W$, and mark the vertex W .*

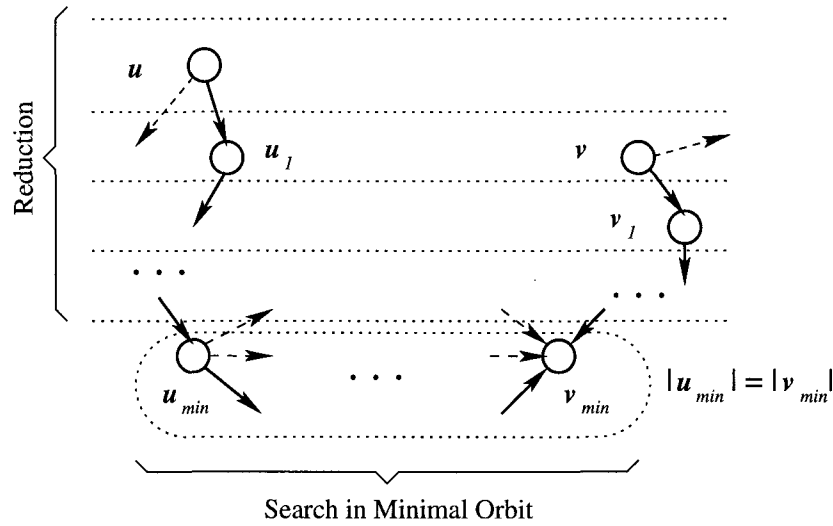


Figure 2.1: Whitehead's Algorithm for two elements u and v . Two stage procedure: computes u_{min} , v_{min} ; enumerates all elements of length $|u_{min}| = |v_{min}|$.

Since $Orb_{min}(U)$ is finite BFS terminates, say in l steps, where

$$l \leq |Orb_{min}(U)| |\Omega_n|$$

It is easy to see that Δ_l is a tree, containing all vertices from $Orb_{min}(U)$. This implies that $V \in Orb_{min}(U)$ if and only if $V \in \Delta_l$. Moreover, the unique path connecting U and V in Δ_l is a shortest path between U and V in $Orb_{min}(U)$, and the sequence of labels along this path is a sequence of Whitehead automorphisms (required in Problem 2.1.5) that connects U and V inside $Orb_{min}(U)$.

From the computational view-point, if memory allows, it is more efficient to start building maximal trees in both graphs $Orb_{min}(U)$ and $Orb_{min}(V)$ simultaneously, until a common vertex occurs.

2.3 Estimates for the time-complexity of the Whitehead's algorithms.

2.3.1 Algorithm DWR.

It is easy to see that transformations of the type 1 in Definition 2.1.4 cannot reduce the total length of a tuple. Hence, to solve Problem 2.1.4 one needs only Whitehead automorphisms of the type 2.

Lemma 2.3.1. *Let F_n be free group of rank n . There are*

$$A_n = 2n4^{(n-1)} - 2n$$

non-trivial Whitehead automorphisms of the type 2.

Proof. Indeed, choose an element $a \in X^{\pm 1}$ then from Definition 2.1.4 every Whitehead automorphism α of the type 2 can be uniquely characterized by a partition of the set $X - \{a\}$, if $a \in X$ (or the set $X - \{a^{-1}\}$ if $a \in X^{-1}$), into 4 subsets $X - \{a\} = A \cup B \cup C \cup D$ such that $x\alpha = x$ if $x \in A$, $x\alpha = a^{-1}x$ if $x \in B$, $x\alpha = xa$ if $x \in C$, $x\alpha = a^{-1}xa$ if $x \in D$. There are $2n$ choices for $a \in X^{\pm 1}$. For a given a there are C_k^{m-1} choices of a subset of $X - \{a\}$ of size k , where C_y^x is the binomial coefficient:

$$C_y^x = \binom{x}{y} = \frac{x!}{y!(x-y)!}.$$

For given $a, A \subset X - \{a\}$ with $|A| = k$ there are C_m^{n-1-k} choices for a subset $B \subset X - \{a\} - A$ of size m . For given $a, A \subset X - \{a\}, |A| = k, B \subset X - \{a\} - A, |B| = m$, there are $C_l^{n-1-k-m}$ choices of $C \subset X - \{a\} - A - B$ of size l . Notice, that there is only one choice for $D = X - \{a\} - A - B - C$. Therefore there are altogether $\sum_{l=0}^{n-1-k-m} C_l^{n-1-k-m} = 2^{n-1-k-m}$ choices for C . Therefore

there are $\sum_{m=0}^{n-1-k} C_m^{n-1-k} 2^{n-1-k-m} = 3^{n-1-k}$ choices for B and C . Therefore there are $\sum_{k=0}^{n-1} C_k^{n-1} 3^{n-1-k} = 4^{n-1}$ choices for A, B, C, D for a given a . Observe, that for a given a the sum above includes also the trivial automorphism (when $A = X - \{a\}$, $B = C = D = \emptyset$). Therefore there are $2n4^{n-1} - 2n$ non-trivial Whitehead automorphisms of the type 2. ■

In the worst-case scenario to perform ELR it requires A_n executions of the following

Procedure 2.3.2 (Substitution Routine SR). *For a given automorphism t of the type 2 make a substitution $x \rightarrow xt$ for each occurrence of each $x \in X^{\pm 1}$ in U , and then make all possible cancellations.*

Since the length of the word xt is at most 3 the time needed to perform this routine is bounded from above by $c|U|$, where c is a constant which does not depend on $|U|$ and the rank of F . Since DWR executes ELR at most $|U|$ times the time-complexity function of DWR is bounded from above by

$$cA_n|U|^2 = c(2n4^{n-1} - 2n)|U|^2,$$

This bound depends exponentially on the rank n of the group $F = F_n(X)$. For example, if $k = 1$, $n = 10$, and $|U| = 100$, the estimated number of steps for DWR is bounded above by

$$c(20 \cdot 4^9 - 20)100^2 > c(5 \cdot 10^{10}).$$

Whether this bound is tight in the worst case is an open question. In any event, computer experiments which we ran on a dual Pentium III, 700 Mhz processor computer with 1Gb memory show (see Table 4.6) that the standard DWR cannot find U_{min} on almost all inputs U which are pseudo-randomly generated primitive elements

of length more than 100 in the group F_{10} , while working non-stop for more than an hour.

The accuracy of the bound depends on how many automorphisms from Ω_n do reduce the length of a given input U . To this end, put

$$LR(U) = \{t \in \Omega_n \mid |Ut| < |U|\}$$

Now, the number of steps that ELR performs on a *worst-case* input U is bounded from above by

$$\max\{A_n - |LR(U)|, 1\}$$

(if the ordering of Ω_n is such that all automorphisms from $LR(U)$ are located at the end of the list $\Omega_n = \{t_1, \dots, t_m\}$).

If we assume that the automorphisms from $LR(U)$ are distributed uniformly in the list Ω_n then DWR needs

$$A'_n = \frac{A_n}{|LR(U)|}$$

steps on average to find an automorphism which reduces length of U .

The results of our experiments (for $k = 1$) indicate that the average value of $|LR(U)|$ for a non-minimal U of the total length l rapidly converges to a constant LR_n when $l \rightarrow \infty$. In Table 2.1 and Figure 2.2 we present values of the $\frac{LR_n}{A_n}$ that occur in our experiments for $k = 1$. This allows us to make the following statement.

Conclusion 2.3.3. *The average number of length reducing Whitehead's automorphisms for a given "generic" non-minimal word $w \in F_n$ does not depend on the length of $|w|$, it depends only on the rank n of the free group F_n (for sufficiently long words w).*

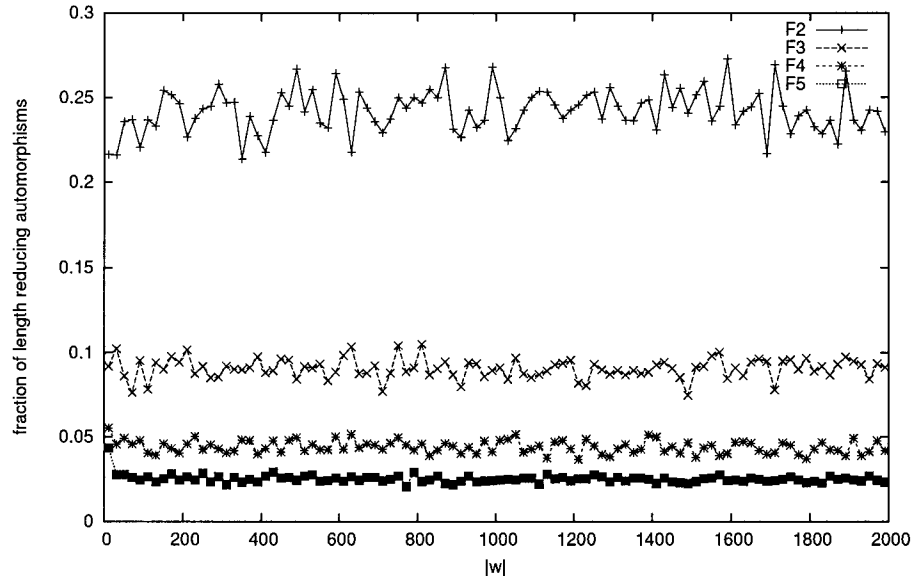


Figure 2.2: Estimates of LR_n/A_n on inputs of various lengths, $n = 2, 3, 4, 5$.

We return to this discussion in Section 2.5, where we define the Whitehead Complexity of an element in F .

$ w $	F_2	F_3	F_4	F_5
0..199	.24	.09	.04	.03
200..599	.24	.09	.05	.03
600..999	.24	.09	.04	.02
1000..1399	.25	.09	.04	.02
1400 ... 1800	.24	.09	.04	.02

Table 2.1: Estimates of LR_n/A_n on inputs of various lengths, $n = 2, 3, 4, 5$.

2.3.2 Algorithm DWMS

The obvious upper bound for the time-complexity of DWMS is much higher, since one has to take into account all Whitehead automorphisms. It is easy to see that

there are

$$B_n = 2n(2n-2)(2n-4) \dots 2 = 2^n(n!)$$

Whitehead automorphisms of the type 1.

To run LS routine on U it requires at most $d(A_n + B_n)$ runs of SR (which has complexity $c|U|$), where d is a constant which does not depend on U and n . Now, to construct $Orb_{min}(U)$ it takes at most $|Orb_{min}(U)|$ runs of LS, hence one can bound the time complexity of DWR from above by

$$d \cdot (A_n + B_n) \cdot c \cdot |U| \cdot |Orb_{min}(U)|.$$

This shows that DWMS may be very slow (in the worst-case) just because there are too many Whitehead automorphisms in the rank n for big n . Moreover, the size of $Orb_{min}(U)$ can make the situation even worse. Obviously the number of elements in $Orb_{min}(U)$ cannot be greater than the total number of tuples of length $|U|$

$$|Orb_{min}(U)| \leq 2n(2n-1)^{|U|-1}, \quad (2.3.1)$$

hence a very rough estimates give the following upper bound for the time-complexity of DWMS

$$d \cdot c \cdot (2n4^{(n-1)} - 2n + 2^n n!) \cdot |U| \cdot 2n(2n-1)^{|U|-1}.$$

One can try to improve on this upper bound through better estimates of $|Orb_{min}(U)|$. It has been shown in [48] that for $k=1$ and $n=2$ the number $|Orb_{min}(U)|$ is bounded from above by a polynomial in $|U_{min}|$. It was also conjectured in [48] that this result holds for arbitrary $n \geq 2$, and for $n=2$ the upper bound is the following:

$$|Orb_{min}(U)| \leq 8|U_{min}|^2 + 40|U_{min}|.$$

Recently, B.Khan proved in [39] that the bound above holds, indeed. Moreover, there are unpublished results [42] which say that the polynomial bounds might hold for all ranks. Still, independently of the size of the set $Orb_{min}(U)$, the number $B_n + A_n$ of elementary Whitehead automorphisms in rank n makes DWMS impractical for sufficiently big n .

The net outcome of the discussion above is that the algorithms DWR and DWMS are intractable for "big" ranks, even though for a fixed rank n DWR is quadratic in $|U|$ and DWMS could be polynomial in $|U|$.

2.4 General Length Reduction Problem.

Observe that the main part of DWR is the elementary length reduction routine ELR, which for a given tuple $U \in F^k$ finds a *Whitehead* automorphism $\varphi \in \Omega(F)$ such that $|U\varphi| < |U|$.

Definition 2.4.1. *An arbitrary automorphism $\varphi \in Aut(F)$ is called length-reducing for $U \in F^k$ if $|U\varphi| < |U|$.*

Whitehead's theorem guarantees that the sequence of length-reducing automorphisms will always lead to a solution to the Whitehead Reduction problem. The main obstacle to solving the problem efficiently is the complexity of ELR procedure (see Picture 2.3). We have seen in Section 2.3 that the time-complexity of the standard Whitehead algorithm for Problem 2.1.4 depends mostly on the cardinality of the set Ω_n which is huge for big n . Obviously, to solve Problem 2.1.4 it suffices to find an arbitrary (not necessary Whitehead) length-reducing automorphism for a non-minimal tuple U . One of the key ideas on improving the efficiency of Whitehead algorithms is

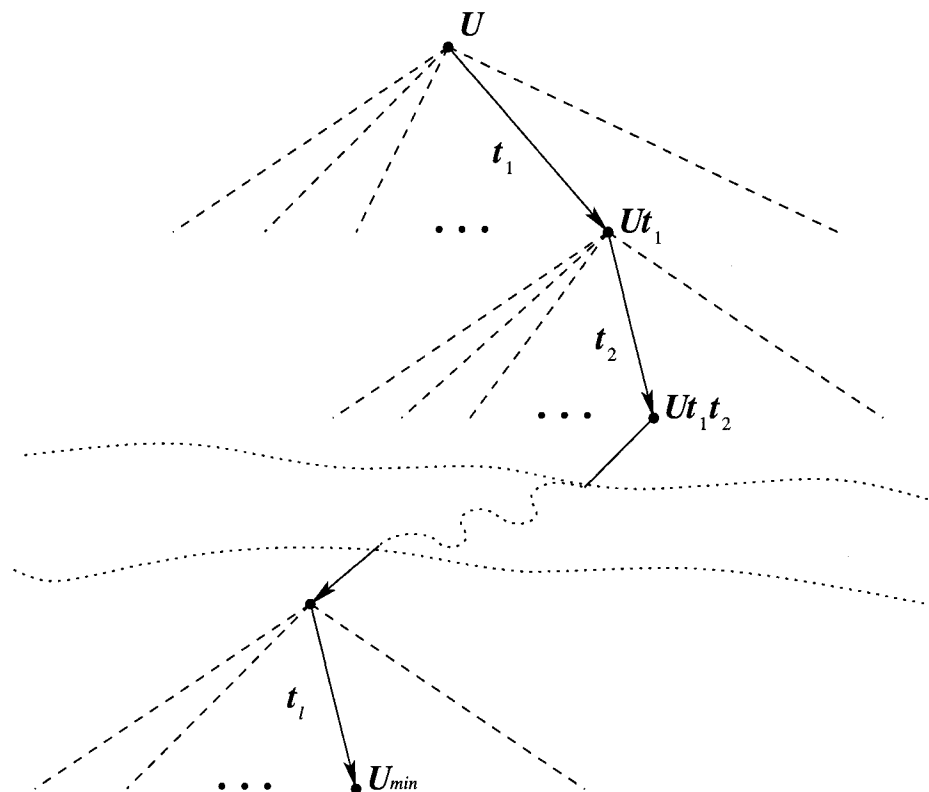


Figure 2.3: Whitehead method for reduction problem. On each iteration a length-reducing automorphism has to be found. This procedure depends exponentially on the rank of a free group and the main obstacle in solving the problem efficiently.

to replace Ω_n by another smaller set of automorphisms of F or to use a different strategy to find length-reducing automorphisms. To this end we formulate the following problem.

Problem 2.4.1 (Length-Reduction Problem LRP). *For a non-minimal tuple $U \in F^k$ find a length-reducing automorphism.*

Theorem 2.1.3 gives one solution to LRP - the algorithm DWR. In case of the free group of rank 2, LRP can be solved using Nielsen set of elementary transformations. Let $\Omega_2 \subset \text{Aut}(F_2)$ be the set of Whitehead automorphisms of type 2 and $N_2 \subset \text{Aut}(F_2)$ be the set of Nielsen automorphisms. The following lemma shows that N_2 is sufficient to solve Length Reduction Problem in F_2 .

Lemma 2.4.2. *None of the automorphisms from the set $\Omega_2 - N_2$ can reduce the length of a cyclically reduced word $w \in F_2$.*

Proof. We have mentioned before that none of the Whitehead automorphisms of type 1 can reduce the length of a word. There are only two Whitehead automorphisms of type 2 in Ω_2 that are not in N_2

$$\Omega_2 - N_2 = \{\varphi_a, \varphi_b\},$$

where $\varphi_a = \begin{pmatrix} a \rightarrow b^{-1}ab \\ b \rightarrow b \end{pmatrix}$ and $\varphi_b = \begin{pmatrix} a \rightarrow a \\ b \rightarrow a^{-1}ba \end{pmatrix}$. We show the proof for automorphism φ_a , i.e. we proof that

$$|\varphi_a(w)| \geq |w|,$$

for any cyclically reduced word $w \in F_2$.

Note, that automorphism φ_a is equivalent to map

$$\varphi'_a = \begin{pmatrix} a \rightarrow b^{-1}ab \\ b \rightarrow b^{-1}bb \end{pmatrix},$$

where b is sent to b written in a non-reduced form. Let $w = x_1 \dots x_l$ be a cyclically reduced word from F_2 . We formally apply map φ'_a :

$$\varphi'_a(w) = \varphi'_a(x_1 \dots x_l) = b^{-1}x_1bb^{-1}x_2b \dots b^{-1}x_lb = b^{-1}x_1 \dots x_lb.$$

Since w is cyclically reduced, we know that $x_1 \neq x_l^{-1}$, therefore

$$|\varphi'_a(w)| = |\varphi_a(w)| \geq |w|$$

for any non-empty cyclically reduced word $w \in F_2$. ■

The cardinality of the Nielsen set N_n

$$|N_n| = 4n(n-1),$$

i.e. it is only a quadratic function of the rank n of a free group, where Ω_n , used in DWR, depends exponentially on n . Obviously it would be advantageous to use a smaller set N_n . Unfortunately, results of Lemma 2.4.2 do not hold for free groups with ranks greater than 2 (see [44]) and DWR remains the only known deterministic solution to LRP.

In Chapter 4 we describe a genetic algorithm which, we believe, solves LRP much more efficiently on average than DWR. In Chapter 6 we describe heuristics designed to speed up the performance of DWR and reduce algorithm's dependence on the rank of a group.

2.5 The Whitehead's complexity

In this section we discuss possible complexity (or size) functions suitable to estimate the time-complexity of different variations of Whitehead algorithms. Below we suggest a new complexity function based on the distance in the Whitehead's automorphism graph.

Let $F = F_r$, $Y \subset \text{Aut}(F)$ a set of generators of the group $\text{Aut}(F)$, $\Gamma(F, Y) = \Gamma(F, 1, Y)$ the automorphism graph on F relative to Y (see Section 2.1). For a word $w \in F$ we define $WC_Y(w)$ as a minimal number of automorphisms from $Y^{\pm 1}$ required to reduce w to a minimal one $w_{\min}$. Notice that $WC_Y(w)$ is the length of a geodesic path in $\Gamma(F, Y)$ from w to some $w_{\min}$. If Y is the set of all Whitehead automorphism Ω_r , then we call $WC_Y(w)$ the *Whitehead's complexity* of w and denote it by $WC(w)$. Similarly, one can introduce the Nielsen's complexity of w , T -complexity, etc. In this context minimal elements have zero Whitehead complexity.

Remark 2.5.1. *The Whitehead's complexity function $WC(w)$ is an adequate complexity function to measure performance of various modifications of the Whitehead's algorithm.*

Indeed, let $\mathcal{K}$ be a class of Whitehead's-type algorithms which use an arbitrary generating set $Y \subset \Omega_r$ of Whitehead automorphisms to find a minimal word $w_{\min}$ for an input word w . The best possible algorithm of this type is the non-deterministic Whitehead algorithm NDWA with an oracle that at each step i gives a length reducing automorphism $t_i \in Y$ such that $|wt_1 \dots t_i| < |wt_1 \dots t_{i-1}|$. Clearly, it takes $WC_Y(w)$ steps for NDWA to produce $w_{\min}$. Thus, measuring efficiency of an algorithm $A \in \mathcal{K}$ in terms of WC_Y gives us a comparison of performance of A to the performance of

the best possible algorithm in the class. ■

Of course, WC complexity is mostly a theoretical tool, since, in general, it is harder to compute $WC(w)$ than to find w_{min} . It follows from the Whitehead's fundamental theorem that $WC(w) \leq |w|$ for every $w \in F$.

In Table 2.2 we collect some experimental results on the relation between $WC(w)$ and $|w|$.

	F_2	F_5	F_{10}	F_{15}	F_{20}
$ wt / w , t \in \Omega$	1.06	1.29	1.38	1.41	1.43
$ wt / w , t \in T$	1.10	1.20	1.12	1.08	1.06

Table 2.2: $WC(w)$ vs $|w|$. Average ratio $|wt|/|w|$ for Whitehead and Restricted Whitehead automorphisms.

This leads to the following

Conclusion 2.5.1. *Let $W_m = \{w \in F_r \mid WC(w) = m\}$. Then there exists a constant c_r such that*

$$|w| \geq c_r^m \cdot |w_{min}|$$

for "most" elements in $w \in W_m$.

We have seen (see Conclusion 2.3.3) that the number of steps required by DWR to solve LPR does not depend on the length of the input tuple. In Section 4.3.1 we present experimental results which show that the Genetic version of the Whitehead method also does not depend on the length of the input tuple when solving the Length-Reducing Problem. These facts suggest that one might prefer a more sensitive "size" functions like Whitehead's complexity $WC(w)$ instead of commonly used functions based on the word's length.

Analyze confidence intervals around mean. Distribution tail information. For example what is the probability of the 3 std deviation tail? Need to add std deviations.

Chapter 3

Experimenting with free groups

In this section we briefly discuss several general problems arising in experiments with infinite groups.

Let $\mathcal{A}$ be an algorithm for computing with elements from a free group $F = F_r(X)$. Suppose that the set of all possible inputs for $\mathcal{A}$ is an infinite subset $S \subset F$. Statistical analysis of experiments with $\mathcal{A}$ involves three basic parts:

1. Creating a finite set of test inputs $S_{test} \subset S$,
2. Running $\mathcal{A}$ on inputs from S_{test} and collecting outputs,
3. Statistical analysis of the resulting data.

A properly designed scientific experiment provides evidence to accept or reject the hypothesis that the algorithm performs in a specified way. To properly set up such an experiment, in a way that it can be repeated and the evidence verified by another researcher, considerable attention must be paid to experimental protocol.

3.1 Experimental Protocol

The experimental protocol states the quantity to be measured, the accuracy to which it is to be measured, the population of algebraic objects on which the decision or classification algorithm is to be applied, and how the tuning parameters of the algorithm are to be set or varied. Then the protocol must give the experimental design and the data analysis plan.

The experimental design must describe how a suitably random, independent, and representative set of objects from the specified population is to be sampled, generated, or acquired. If the population includes, for example, words with a range of lengths of interest or these words have a variety of specific properties, then the sampling mechanism must assume that a reasonable number of words are sampled with the words appearing in each of different lengths or appearing with properties throughout their permissible range.

In general, the essential variation of the objects in the population can be described by some number, say N variables. If the N variables $X_1, \dots, X_N$ having to do with object properties like length, order, equivalence class etc., have respective range sets $R_1, \dots, R_N$, then the sampling design must assure that objects sample the domain $R_1 \times R_2 \times \dots \times R_N$ in a representative way. The number of samples may only be a small fraction of the number of possibilities in $R_1 \times R_2 \times \dots \times R_N$. This suggests that the experimental design may have to make judicious use of a latin square layout [12]. If there are some free parameters that must be set in the algorithm, then these parameters can as well be part of the N variables.

The experiments must then be carried out for each object in the sample. Statistical analysis of the results depends on the features that are going to be tested (average

running time of the algorithm, expected frequencies of outputs of a given type, etc.). Thus, estimation of the running time of an algorithm depends on how we measure "complexity" or "size" of the inputs. For example, it turns out that the running time of the Stochastic version of Whitehead algorithm does not depend essentially on the length of an input word s , so it would be meaningless to measure the time complexity of DWR in terms of the length of s (as is customary in computer science).

Suppose there are M different measurements $\hat{y}_1, \dots, \hat{y}_M$ obtained when an algorithm is applied to particular object in the sample. As a result of these measurements, which are really statistical inferences, there will be a difference between the true values $y_1, \dots, y_M$ of the measured quantities and the measured values themselves. The accuracy criterion must state how the comparison between the true values and the measured values will be evaluated. Finally, the experimental data analysis plan must state how the hypothesis that the algorithm meets the specified requirement will be tested. It indicates how the observed data (the true values and the corresponding measured values) will be analyzed. The plan must be supported by a theoretically developed statistical analysis which shows that an experiment carried out according to the experimental design and analyzed according to the data analysis plan will produce a statistical test itself having a given accuracy. That is, since the entire population was only sampled, the sampling fluctuation will introduce random fluctuation in the test results. For some fraction of experiments carried out according to the protocol, the hypothesis to be tested will be accepted but if the algorithms were to be tried on the complete population of object variations, it would not meet the specified requirements; and for some fraction of experiments carried out according to the protocol, the hypothesis to be tested will be rejected but if the algorithm were to

be tried on the complete population of object variations, it would meet the specified requirements.

The specified size of these errors of false acceptance and missed acceptance will dictate the number of objects to be in the sample. This relation between sample size and false acceptance rate and missed acceptance rate of the test for the hypothesis must be determined on the basis of statistical theory. One would certainly expect that the sample size would be large enough so that these error rates would be below 20%.

In summary, any experimental protocol must include the following items.

1. The protocol must state:
 - what is to be measured on each experimental trial
 - what is to be inferred (estimated) from the measurements or what hypothesis is to be tested (the requirement)
 - what is the theoretically expected behavior of what is to be estimated
 - with what precision (standard deviation) is the inferred quantity to be estimated or with what misdetect and false alarm error is the hypothesis to be tested
 - over what population of algebraic objects and algorithm tuning parameters are the experiments to be done
2. The protocol must layout an experimental design which describes
 - how a suitable random, independent, and representative set of objects from the specified population is to be sampled, generated, or acquired, including the setting of all parameters relevant to the population

- what parameters of the algorithm will be varied and how their values will be varied
- what parameters of the algorithm will be set and what values they will be set at
- the experiments which must be carried out for each object in the population
- the accuracy criterion which states how the comparison between the true values and the measured values will be evaluated

3. The protocol must have a data analysis plan which

- states how to statistically compare the behavior of what is estimated and the theoretically expected behavior of what is estimated to verify the hypothesis that the observed behavior is consistent with the theoretical expectations
- states how to test the hypothesis that the algorithm meets the specified requirement
- indicates how the data (the true values and the corresponding measured values) will be analyzed
- tells explicitly what performance curves will be generated

4. The data analysis plan

- must be supported by a theoretically developed statistical analysis
- and show that an experiment carried out according to the experimental design and analyzed according to the data analysis plan will produce results

having a given accuracy (i.e., the rates of false alarms and miss detections due to the experimental sampling variation meet the stated requirement)

Below we suggest some particular ways to approach all these problems in the case of the Whitehead's algorithms.

3.2 Data generation

3.2.1 Generating “random” elements in a free group

The notion of a random element in F depends on a chosen measure on F . Since F is infinite, elements in F are not uniformly distributed. The problem cannot be solved just by replacing F with a finite ball B_n , of all elements in F of length at most n , for a big number n . Indeed, firstly, the ball B_n is too big for any practical computations; secondly, from group-theoretic view-point elements in B_n usually are not uniformly distributed. We refer to [9] and [8] for a thorough discussion of this matter.

It seems that the most obvious choice for the set S_{test} to test performance of various Whitehead algorithms would be a finite set S_F of randomly chosen elements from F . It turns out, that this choice is not good at all since with a high probability a random element in F is already minimal [38] (see Figure 3.2). Nevertheless, the set S_F plays an important part in the sequel as a base for other constructions.

Let $X^{\pm 1} \in G$ be a generating set of a group G .

Definition 3.2.1. *A Cayley graph $C(X, G)$ of the group G is a graph having one vertex associated with each element in G and edges (g, h) if and only if $gh^{-1} \in X^{\pm 1}$.*

The Cayley graph may depend on the choice of a generating set, and is always

connected. Obviously, Cayley graphs of infinite groups are infinite. Fragments of Cayley graphs of the free group F_2 and infinite abelian group $Z \times Z$ are given in Figure 3.1.

A random element w in $F = F_r(X)$ can be produced as the result of a no-return simple random walk on the Cayley graph of F with respect to the set of generators X (see [8] for details). In practice this amounts to a pseudo-random choice of a number l (the length of w), and a pseudo-random sequence $y_1, \dots, y_l$ of elements $y_i \in X^{\pm 1}$ such that $y_i \neq y_{i+1}^{-1}$, where y_1 is chosen randomly from $X^{\pm 1}$ with probability $1/2r$, and all others are chosen randomly with probability $1/(2r - 1)$. It is convenient to structure the set S_F as follows:

$$S_F = \bigcup_{l=1}^L S_{F,l}, \quad S_{F,l} = \bigcup_{i=1}^K w_{i,l}$$

where $w_{i,l}$ is a random word of length l and L, K are parameters.

To show that S_F is not suitable for testing Whitehead algorithms we provide experimental evidence that even for practical input sizes ($l = 1000$) minimal elements dominate in S_F .

To find all minimal elements in S_F we run the standard deterministic Whitehead algorithm DWR on every $s \in S_F$. Since DWR is very slow for big ranks we experimented with free groups $F = F_r$ for $r = 3, 4, 5$. In Figure 3.2 we present the fractions of minimal elements among all elements of a given length in S_F .

This experimental data leads to the following statement.

Conclusion 3.2.1. *Starting with length 200, almost all elements in $F_r, r \geq 2$ are Whitehead minimal.*

We refer to [38] for a rigorous formulation of the corresponding mathematical statement.

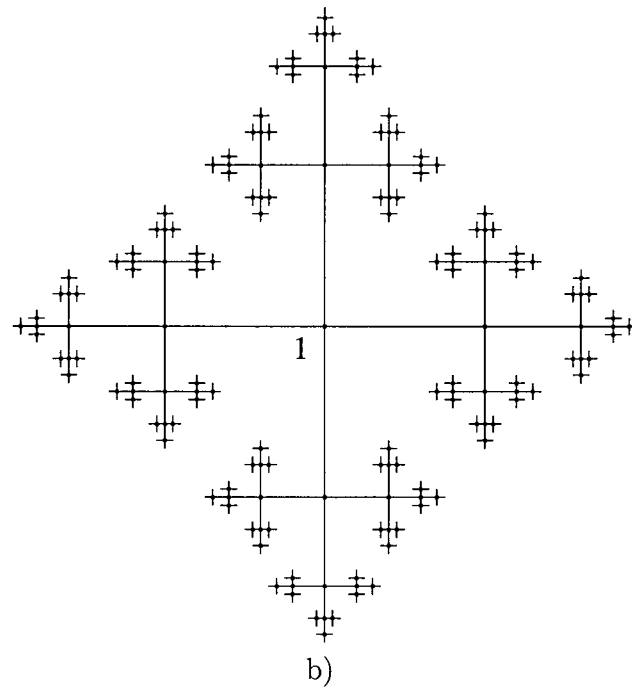
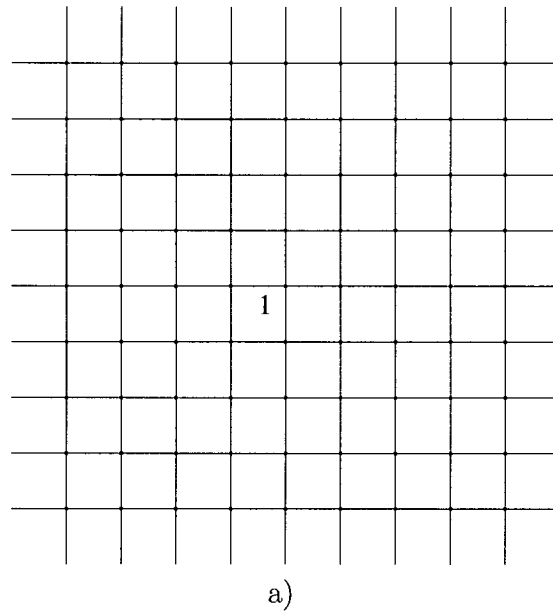


Figure 3.1: Fragments of Cayley graphs: a) infinite abelian group on two generators $\mathbb{Z} \times \mathbb{Z}$; b) free group on two generators F_2 . Vertices represent elements of the corresponding group.

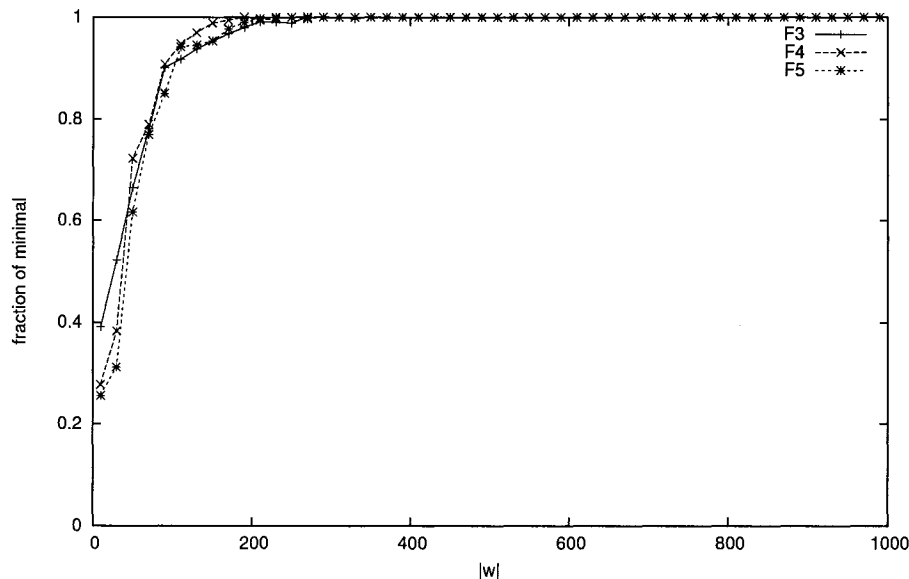


Figure 3.2: Fractions of Whitehead-minimal elements in a free group F_r , $r = 3, 4, 5$. Almost all elements of length greater than 200 are Whitehead minimal.

The running time $T_{DWR}(w)$ of the standard Whitehead algorithm DWR on a minimal input w is very easy to estimate. Indeed, in this case DWR applies the substitution routine SR for every Whitehead automorphism of the second type. Since there are A_r such automorphisms (see Section 2.2), then

$$A_r \leq T_{DWR}(w) \leq c \cdot A_r |w|.$$

The time spent by the stochastic algorithm SWR on a random input w depends solely on the built-in termination condition: if it is heuristic (see Section 4.2.3), then SWR stops after $pE(w)$ iterations, where $E(w)$ is the expected running time for SWR on the input w and p is a constant parameter; if it is deterministic then again it takes A_r steps for SWR to halt. This shows that the set S_F does not really test how SWR works, instead, it tests only the termination conditions.

We summarize the discussion above in the following statement.

Conclusion 3.2.2. *The time-complexity of Whitehead algorithms DWR and SWR on generic inputs from S_F is easy to estimate. The set S_F does not provide any means to compare algorithms DWR and SWR.*

It follows that one has to test Whitehead algorithms on inputs $w \in F$ which are *non-minimal*.

3.2.2 Generating “random” automorphisms of a free group

One way to randomly generate a non-minimal element in F is by transforming a random element with some automorphism from $Aut(F)$. In this section we describe a way to generate “random” automorphisms of a free group.

In general, a random automorphism φ with respect to a fixed finite set A of generators of the group $Aut(F)$ (for example the set of elementary Whitehead automorphisms Ω , or Nielsen set N) can be generated as the result of a no-return simple random walk on the Cayley graph $\Gamma(Aut(F), A)$ of $Aut(F)$ with respect to the set of generators A . Unfortunately, the structure of $\Gamma(Aut(F), A)$ is very complex, and it is not clear how to simulate such a random walk in practice. In theory, since the word problem for $Aut(F)$ is decidable, we could construct any finite ball B_n of radius n in $\Gamma(Aut(F), A)$. But, in practice, for sufficiently large n the B_n is too big to be tractable for modern computations.

Instead, we elected to make a no-return random walk on the Cayley graph $\Gamma(F(\Omega_n), \Omega_n)$ of a free group with the basis Ω_n , as was described in Section 3.2.1. This choice is based tacitly on the following assumption: *a random automorphism φ can be generated via a no-return simple walk on $\Gamma(F(\Omega_n), \Omega_n)$.* It is not altogether clear whether

this is a reasonable assumption or not, but this is the only one which known to be practical. Now, a random automorphism $\varphi \in \text{Aut}(F)$ is a freely reduced product $\varphi = t_1 \dots t_l$ of l randomly uniformly chosen automorphisms from Ω_n with $t_i \neq t_{i+1}^{-1}$. The number $l = l(\varphi)$ is called the *length* of φ .

3.2.3 Generating “random” non-minimal elements in a free group

To this end we describe how one can generate a finite set $S_{NMin}(r)$ of non-minimal elements in a free group F_r by applying random automorphisms to elements from S_F . More precisely, put

$$S_{NMin}(r) = \bigcup_l \bigcup_{1 \leq i \leq K} w_{i,l} \varphi_i,$$

where $\varphi_i \in \text{Aut}(F)$ is a randomly generated automorphism, $w_{i,l} \in S_F$ with $|w_{i,l}| < |w_{i,l} \varphi_i|$. Since almost all elements from S_F are minimal it is easy to generate a set like $S_{NMin}(r)$.

We would like to distinguish two specific types of non-minimal elements that will play important role in the following chapters. First, we define a set of non-minimal elements at distance 1 from minimal ones, elements of Whitehead complexity 1:

$$S_{WC1}(r) = \bigcup_l \bigcup_{1 \leq i \leq K} w_{i,l} \varphi_i,$$

where φ_i and $w_{i,l}$ defined as above, but $l(\varphi) = 1$.

The second set S_P contains randomly generated primitive elements in F (see Definition 2.1.6)

$$S_P(r) = \bigcup_{1 \leq i \leq K} x_i \varphi_i,$$

where φ_i is a random automorphism and x_i is chosen uniformly randomly from $X^{\pm 1}$.

3.3 Minimal sample size estimation.

Suppose we perform an experiment where results are based on some measure $\mathcal{Y}$. One of the basic questions that every researcher has to answer is: how many samples do we need to generate in order for our results to be reasonably reliable. If the underlying distribution of the sampled data is known, then this question can be answered by directly comparing the distribution of the sample to the true population distribution. There are known parametric and non-parametric tests designed for such purposes. However, we are mostly interested in approaches which do not assume knowledge of the underlying distribution.

In this section we give two methods which estimate the size of the sampled dataset. The first approach is applicable whenever the observed measure can result in only two possible values. The second is more general and based on the difference between the sample mean and the corresponding true mean of the population.

3.3.1 Estimating the minimal size of a sample set of boolean observations.

Let $Y = \langle y_1, \dots, y_N \rangle$ be a vector of independent identically distributed observations such that $y_i \in \{0, 1\}$ and let K be the number of "successes", i.e. $K = \sum_{i=1}^N y_i$. Denote the fraction of 1's in Y

$$\hat{p}_N = \frac{K}{N}.$$

Note, that $\hat{p}_N$ is an estimate for the probability of a measurement y to be 1 based on the sample set of size N . Let $p = \text{Prob}(y = 1)$ be the true probability of the "success". We state our problem of finding the minimal size N of a sample set as

that of finding the minimal N such that

$$Prob\{|p - \hat{p}_N| < \epsilon \mid K, N\} \geq 1 - \alpha, \quad (3.3.1)$$

where ϵ is the given minimal allowed divergence from the true probability and α is a significance parameter. Inequality 3.3.1 can be rewritten as

$$Prob\{\hat{p} - \epsilon < p < \hat{p} + \epsilon \mid K, N\} \geq 1 - \alpha.$$

Assuming a continuous density function we can restate the problem as that of finding the minimal N such that

$$\int_{\hat{p}+\epsilon}^{\hat{p}-\epsilon} P(p|K, N) dp \geq 1 - \alpha.$$

Using Bayes rule

$$\begin{aligned} P(p|K, N) &= \frac{P(p, K, N)}{P(K, N)} \\ &= \frac{P(K|p, N)P(p, N)}{\int_0^1 P(K, N, p) dp} \\ &= \frac{P(K|p, N)P(p, N)}{\int_0^1 P(K|N, p)P(p, N) dp} \end{aligned} \quad (3.3.2)$$

The true probability p should not depend on the number of samples in Y , therefore $P(p, N) = P(p)P(N)$. The prior probabilities $P(p)$ are assumed to be uniform. In such cases the value of $P(p)$ does not depend on p and can be moved out of the integral. Now, recall that the probability of K "successes" in N samples follows the Binomial distribution and

$$P(K|p, N) = \binom{N}{K} p^K (1-p)^{N-K}. \quad (3.3.3)$$

Using the independence assumption and equations 3.3.2 and 3.3.3 we have

$$\begin{aligned}
P(p|K, N) &= \frac{P(K|N, p)P(p)P(N)}{\int_0^1 P(K|N, p)P(p)P(N)dp} \\
&= \frac{P(K|p, N)}{\int_0^1 P(K|N, p)dp} \\
&= \frac{\binom{N}{K} p^K (1-p)^{N-K}}{\int_0^1 \binom{N}{K} p^K (1-p)^{N-K} dp} \\
&= \frac{p^K (1-p)^{N-K}}{\int_0^1 p^K (1-p)^{N-K} dp} \\
&= \frac{1}{B(K+1, N-K+1)} p^K (1-p)^{N-K}. \tag{3.3.4}
\end{aligned}$$

Now the problem is reduced to finding the minimal N such that

$$\frac{1}{B(K+1, N-K+1)} \int_{\hat{p}-\epsilon}^{\hat{p}+\epsilon} p^K (1-p)^{N-K} dp \geq 1 - \alpha.$$

This is an incomplete beta function. Procedures for computing beta distributions are available in standard libraries, `cdcdfib` [14] for example. To find N , one can start with a small number of samples and repeatedly increase N until the condition above holds.

3.3.2 Estimating the minimal size of a set of real valued observations.

Let $Y = \langle y_1, \dots, y_N \rangle$ be the set of independent identically distributed observations $y_i \in \mathbb{R}$. Let $\bar{y}_N = \sum_{i=1}^N y_i / N$ be the mean of the sample of size N and μ be the true but unknown mean.

Following the idea introduced in the previous section, we define the problem of

finding the minimal size of the dataset Y as that of finding the minimal N such that

$$Prob\{|\bar{y}_N - \mu| < \epsilon\} \geq 1 - \alpha,$$

i.e. we would like the sample mean to be different from the true mean by not more than a given ϵ .

For the large enough N ($N > 100$), using results following from the Central Limit Theorem [24],

$$\bar{y} \sim \mathcal{N}(\mu, \frac{\sigma^2}{N}),$$

where $\mathcal{N}(\mu, \frac{\sigma^2}{N})$ denotes normal distribution with mean μ , variance σ^2/N and σ^2 is the unknown true variance of observations. Subtracting μ we have

$$\bar{y} - \mu \sim \mathcal{N}(0, \frac{\sigma^2}{N})$$

and

$$\frac{\bar{y} - \mu}{\sigma/\sqrt{N}} \sim \mathcal{N}(0, 1),$$

i.e. $\frac{\bar{y} - \mu}{\sigma/\sqrt{N}}$ is a standard normal variate.

Recall the Student's t distribution with $n - 1$ degrees of freedom

$$t_{n-1} = \frac{Z}{V/\sqrt{n}},$$

where $Z \sim \mathcal{N}(0, 1)$, V is a χ^2 distributed with $n - 1$ degrees of freedom and Z, V are independent.

It can be shown that

$$\frac{\bar{y} - \mu}{S/\sqrt{N}} \sim t_{N-1},$$

where $S^2 = \sum_{i=1}^N (y_i - \bar{y})^2 / (N - 1)$ is the sample variance.

Now

$$\begin{aligned}
 \text{Prob}\{|\bar{y} - \mu| < \epsilon\} &= \text{Prob}\{-\epsilon < \bar{x} - \mu < \epsilon\} \\
 &= \text{Prob}\left\{\frac{-\epsilon}{S/\sqrt{N}} < \frac{\bar{y} - \mu}{S/\sqrt{N}} < \frac{\epsilon}{S/\sqrt{N}}\right\} \\
 &= \int_{\frac{-\epsilon\sqrt{N}}{S}}^{\frac{\epsilon\sqrt{N}}{S}} t_{N-1}(y) dy
 \end{aligned} \tag{3.3.5}$$

The problem has been reduced to finding the minimal N such that

$$\int_{\frac{-\epsilon\sqrt{N}}{S}}^{\frac{\epsilon\sqrt{N}}{S}} t_{N-1}(y) dy \geq 1 - \alpha.$$

3.4 Testing the Consistency

One of the important issues in experimental research is the ensuring of representativeness and unbiasedness of the studied data. Classical statistical methods employ hypothesis testing techniques which might justify that the sampled data, indeed, come from the true distribution of the population. This requires description of the true distribution (parametric or non-parametric). Unfortunately, there is very little known about distributions of algebraic objects in free groups. Moreover, most of the known results can not be applied for practical reasons.

Below we describe several methods designed to estimate parameters and, to some extend, to detect unexpected biases in the procedures which generate random elements in $F(X)$.

3.4.1 Even/odd length ratio

The first example of testing the consistency of the random element generator is based on the assumption that in the sample Y of N randomly generated elements from a

free group F , the number of elements of odd lengths should be relatively equal to the number of elements of even lengths. We assume that elements in Y are independently generated. Let K be the true number of even length elements in sample of size N and K_0 be the observed number of even length elements in the sample Y . K follows the binomial distribution and

$$Prob(K|N, p) = \binom{N}{K} p^K (1-p)^{N-K}, \quad (3.4.1)$$

where p is the true probability of success and under the assumption above, $p = 0.5$.

Now, let $\hat{p}$ be the probability of a randomly generated word w having an even length. We should be able to test the following hypothesis:

$$H_0 : \hat{p} = 0.5.$$

We would like to reject H_0 if K_0 is significantly larger or smaller than $K_L(\alpha)$ and $K_S(\alpha)$ respectively, where $K_L(\alpha)$ is the smallest integer such that

$$\sum_{K_L}^N \binom{N}{K} p^K (1-p)^{N-K} \leq \alpha/2,$$

$K_S(\alpha)$ is the largest integer such that

$$\sum_0^{K_S} \binom{N}{K} p^K (1-p)^{N-K} \leq \alpha/2$$

and α is the significance level of the test. Finally, we do not reject the hypothesis if

$$K_0 \in [K_S(\alpha), K_L(\alpha)]$$

for a given α and reject it otherwise.

The values of K_S , K_L become difficult to compute when N is sufficiently large. In such case sum in 3.4.1 can be computed by reducing the binomial distribution to the cumulative incomplete beta distribution [1], [14].

3.4.2 Counting the Generators

Another reasonable assumption that we can make is that for a large enough random word $w \in F(X)$, the generators from X should occur with equal frequencies. For a randomly generated word w we compute the χ^2 statistic

$$\chi^2(w) = \sum_{i=1}^{|X|} \frac{(n_i - E_i)^2}{E_i},$$

where n_i is the number of times generator $x_i^{\pm 1}$ occurs in w and $E_i = |w|/|X|$ is the expected number of occurrences. The statistic $\chi^2(w)$ will approximately follow the chi-square distribution with $|X| - 1$ degrees of freedom (see Section 1.6.1), given the generators, indeed, occur with equal frequency.

To test the consistency of the generating procedure, we construct a vector $Y = \langle \chi^2(w_1), \dots, \chi^2(w_N) \rangle$, where w_i is a randomly generated element in F . Now we can use the Kolmogorov-Smirnov test [16] or, its improvement, the Anderson-Darling test [60] to check the hypothesis that values $y \in Y$ indeed come from the $\chi^2_{|X|-1}$ distribution. Rejection of the hypothesis might indicate an inconsistency in the generated data.

3.4.3 Test based on the Whitehead Graph

Here we extend the method described above to the counts of subwords of length 2 in a word w . The so-called Whitehead Graph plays an important role in analysis of automorphic properties of elements in free groups.

Definition 3.4.1 (Whitehead Labelled Graph). *Let $w \in F(X)$ be a cyclic word. A Whitehead Graph $WG(w) = (V, E)$ is an undirected weighted graph, where the set of vertices V corresponds to generators in $X^{\pm}$ and there is an edge $(x_i, x_j) \in E$ if*

subwords $x_i x_j^{-1}$ or $x_j x_i^{-1}$ occur in w . Every edge (x_i, x_j) is assigned a real valued weight $l_{ij} = n_{ij}/|w|$, where n_{ij} is the number of times the subword $(x_i x_j^{-1})^{\pm 1}$ occurs in w .

The test we describe below is based on the assumption that the weights of a graph $WG(w)$ will be uniformly distributed for a long enough randomly generated word w in F_n [38]. There are total of $4n^2 - 2n$ possible edges in a Whitehead Graph of an element in the free group of rank n . The expected value of each label under the assumption above is

$$E[l_{ij}] = 1/(4n^2 - 2n).$$

We compute the chi-square statistic

$$\chi^2(WG(w)) = \sum_{\forall (x_i, x_j) \in E} \frac{(l_{ij} - E[l_{ij}])^2}{E[l_{ij}]}$$

which approximately follows the chi-square distribution with $4n^2 - 2n - 1$ degrees of freedom.

Similar to the previous example we construct the set $Y = \langle \chi^2(WG(w_1)), \dots, \chi^2(WG(w_N)) \rangle$ of chi-square statistics computed for the randomly generated elements in F_n and perform the Anderson-Darling test to check the hypothesis that the sample Y , indeed, comes from the $\chi^2_{4n^2-2n-1}$ distribution. Rejection of the hypothesis will indicate that the weights of the Whitehead Graph of the randomly generated elements are not uniformly distributed and, therefore, the generating procedure is biased.

3.4.4 Invariance under permutations of the generators

All the tests described above are based on the assumptions which are reasonable only for randomly generated elements that are not restricted to any subsets of a free

group. However, in Section 3.2.1 we argued that random elements are not interesting for the investigation of the Whitehead-Minimization problem. In fact, in most cases we would like to consider elements from a particular subset of a free group. Obviously the assumptions that we are allowed to make will depend on the properties of the elements of interest. Unfortunately, the uniformity assumptions used in the previous examples almost always will be incorrect for a given subset of elements of a free group.

Here we describe a method which tests the unbiasedness of a generating procedure G , using assumption that word properties are invariant under certain permutations of generators in $X^\pm$. This is a very general assumption and can be made almost everywhere except for a very special cases.

Let ρ_i be a permutation of generators in $X^\pm$ with a restriction that if $x_i \leftrightarrow x_j$ then $x_i^{-1} \leftrightarrow x_j^{-1}$, $x_i, x_j \in X^\pm$ and let P denote the total number of such permutations in the consideration. Notice, that for every permutation ρ_i there is a corresponding Whitehead automorphism $\varphi_i \in \Omega$ of type 1. We generate P samples of randomly generated elements in F

$$S_i = \langle w_1^i, \dots, w_{n_i}^i \rangle$$

and compute vectors

$$Y_i = \langle y_1^i, \dots, y_{n_i}^i \rangle,$$

where y_j^i corresponds to the vector of weights of the graph $WG(\varphi_i(w_j^i))$, i.e. Y_i contains vectors of weights of randomly generated elements after permutation ρ_i .

Under the invariance assumption, we should expect the mean vectors of the sets Y_i be relatively equal. Therefore, the consistency check in this case can be reduced to testing of the following hypothesis

$$H_0 : \mu_1 = \mu_2 = \dots = \mu_P,$$

where μ_i is the mean vector of the set Y_i . This is the case of the k -class multivariate test described in Section 1.6.2.

In fact we can use the Bonferroni's approach which significantly simplifies the computations. Let $M = 4n^2 - 2n$ be the number of edges in a Whitehead Graph of a word in the free group F_n . Denote by μ_{ik} the k th component in the mean vector μ_i . It is safe to assume that under the invariance assumption, the mean values of the weights of a given edge in Whitehead Graphs $WG(\varphi_i(w_j^i))$ should not be significantly different for $i = 1, \dots, P$.

The new null hypothesis is

$$H_0 : \mu_{1k} = \mu_{2k} = \dots = \mu_{Pk}, \quad k = 1, \dots, M$$

and the alternative

$$H_1 : \exists i, j, k \text{ s.t. } \mu_{ik} \neq \mu_{jk}.$$

We compute the variance ratio statistic W from the equation 1.6.2 for each edge weight in the Whitehead Graph separately

$$W^* = \langle W_1, \dots, W_M \rangle,$$

where W_k is the statistic W computed on the k th component of the vectors y_j^i , $i = 1, \dots, P$. Now, let

$$p = \langle p_1, \dots, p_M \rangle$$

be the corresponding p -values of the tests. Following the Bonferroni's formula 1.6.3 we reject hypothesis if

$$\min\{p_1, \dots, p_M\} < \alpha/M,$$

where α is the significance level of the test.

Significantly different mean values of the Whitehead Graph weights under the permutations of X will result in the rejection of the hypothesis H_0 , which might indicate a problem in the data generating procedure.

3.5 Experimental Comparison of the Algorithm Performance

It was mentioned earlier that many problems dealing with infinite groups are hard or even unsolvable. The complexity analysis of methods designed to solve or partially solve such problems often deemed to be “impractical”. There are many problems, where situations corresponding to the worst cases do not occur on the major number of problem’s inputs. In practice, it is often possible to restrict inputs such that the worst cases do not occur or occur with small probability. Some optimization techniques might improve on the time required to solve a problem by producing not the optimal but near optimal solutions. The need in the non-classical approach to the complexity analysis of the algorithms resulted in appearing of the *Average Case Complexity* ([43], [66]) where problems characterized not by its worst case behavior but by the behavior averaged over all possible inputs to the problem. Recently, the *Generic Case Complexity* was introduced [8], [38]. In generic case complexity problem is analyzed in accordance with some measure defined on all inputs. The main distinction of such approach is that uncommon inputs might have asymptotic measure 0, which allows one to analyze the behavior of undecidable problems.

In this section we introduce protocols which will allow one to experimentally compare the performance of algorithms. Such approach can be useful in cases when the

theoretical analysis is complicated by the nature of algorithms or even the problem itself. For example, it is often difficult to give reasonable theoretical bounds for complexity of heuristic or stochastic search methods. Moreover, it is common situation when such methods stop only when the solution, indeed, exists. Similar problems arise when the evaluation needs to be done on a restricted subset of the possible inputs and the subset does not have the appropriate theoretical description.

The experimental approach based on the comparison of the algorithms, executed on the identical input sequences. We make an assumption that inputs are fairly represented regarding the compared algorithms. Below we give two measures which allow experimental evaluation and comparison of algorithms' performance even when algorithms are not required to stop on every input.

3.5.1 Experimental "Generic" behaviour

Let $\mathcal{A}$ be an algorithm and $D = \langle (x_i, t_i) \mid i = 1, \dots, N \rangle$ be the results of experimental runs of $\mathcal{A}$ on a given list of inputs, where x_i is the size of the i th input and t_i is the corresponding value of the performance measure, time for example. Notice, that this is not a one-to-one correspondence. Values of t could be different for the same values of x .

Assuming that we know that t_i has polynomial dependency on x_i , we would like to find a degree m of polynomial so that $\text{Prob}\{t > x^m + C\} < \alpha$ for some value of α .

Consider data in the log space. The same statement can be rewritten as: find m, C such that

$$\text{Prob}\{\log t > \log C + m \log x\} < \alpha.$$

Notice that C and m are just the shift and slope of a line in the log space (see

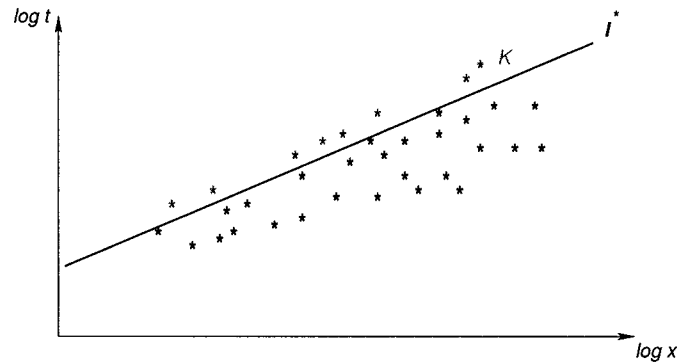


Figure 3.3: Optimal line l^* in the log space with K points above the line.

Figure 3.3). The task can be reduced to a problem of finding proper parameters of the optimal line l^* .

First, we have to decide on the number K of observations (x_i, t_i) for which we allow

$$t_i > x_i^m + C,$$

i.e. K is the number of points above l^* in the log space. Let p be the probability of a point lying above the line l^* . We define K as the largest integer such that

$$\text{Prob}\{p < \epsilon | N, K\} \geq 1 - \alpha,$$

where ϵ is a parameter defining the maximal allowed probability of an observation to be above the line l^* and α is the confidence parameter. Obviously we would like ϵ and α to be small.

To find K observe that K is binomially distributed

$$P(K|N, p) = \binom{N}{K} p^K (1-p)^{N-K}.$$

Using the derivations similar to ones given in Section 3.3.1 we can show that

$$\text{Prob}\{p < \epsilon | N, K\} = \frac{1}{B(K+1, N-K+1)} \int_{p=0}^{\epsilon} p^K (1-p)^{N-K} dp.$$

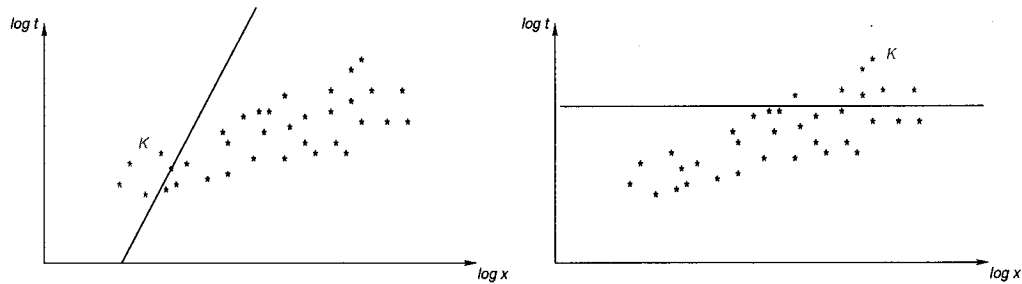


Figure 3.4: Examples of misrepresenting lines.

Therefore, we need to find the largest K s.t.

$$\frac{1}{B(K+1, N-K+1)} \int_{p=0}^{\epsilon} p^K (1-p)^{N-K} dp \geq 1 - \alpha.$$

Given the value K we need to find the optimal line l^* s.t. no more than K points lie above l^* . Moreover, these points should be more or less uniformly distributed along the line and the line should follow the direction of the data. If we do not follow these requirements we might end up with “misrepresenting” lines (see Figure 3.4).

Now our problem can be stated formally as optimization problem.

Problem 3.5.1. Find m and C such that error:

$$E = \frac{\sum_i (\log t_i - \log C - m \log x_i)^q}{V[\log t_i | \log C + m \log x_i < \log t_i]},$$

is minimal and no more than K points lie above the line $m \log x + \log C$. In the equation above V is the variance of $\log x$ coordinate of points lying above the line.

Finally, two algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ can be compared to each other by comparing the corresponding values of m and C . If both algorithms have the same degree m then we can assume that they have the same asymptotic time-complexity.

Useful information can be obtained if we plot values of E against the corresponding values of degree m . If the graph has a global minimum then we should be able to

find the optimal value for m . However, if the graph is flat or has several minimal points then we cannot be certain about the value of m . In this case we either do not have enough data to represent the whole population, or our assumption about the polynomial behavior of the function was not correct.

3.5.2 Counting the Successful Runs

In the case when termination conditions of some or even all algorithms in consideration are not defined on all possible inputs, the straightforward comparison of the execution time will fail because sooner or later one of the algorithms will fall into infinite loop. Here we describe a simple method of comparison of algorithms executed on a given list of inputs.

Let L be the list of inputs, $\mathcal{A}$ be an algorithm and $T(\mathcal{A}, x) \in \mathbb{R}_+$ be the time measure representing the amount of time spent by $\mathcal{A}$ on the input $x \in L$. Notice that T is not bounded from above. Time measure could be defined in any convenient way: number of executions of particular steps of the algorithm, amount of the CPU time, actual time, etc. Of course, if the time measure used to evaluate two different algorithms has to be meaningful and computable for both of them.

Define by $\mathbb{T}$ the total amount of time that we allow for an algorithm evaluation and δ be the maximal time allowed to be spent by an algorithm on a particular input from L . Let $L_k = \langle x_1, \dots, x_k \rangle$ be the initial segment of k elements in the list L and

$$T(\mathcal{A}, L_k) = \sum_{x_i \in L_k} \min(T(\mathcal{A}, x_i), \delta)$$

be the actual time spent by algorithm $\mathcal{A}$ on all inputs in L_k . Define

$$L^*(\mathcal{A}) = \min_k \{T(\mathcal{A}, L_k) \geq \mathbb{T} \mid k = 1, \dots, |L|\}$$

as the smallest initial segment of inputs L such that the total time spent by $\mathcal{A}$ on L_k is at least T . Now, let $F(\mathcal{A}) \subseteq L^*(\mathcal{A})$ be the set of inputs on which algorithm $\mathcal{A}$ failed to produce an answer in time δ .

The performance measure of an algorithm $\mathcal{A}$ on the list of inputs L

$$\mathcal{E} = |L^*(\mathcal{A})| - |F(\mathcal{A})|.$$

The greater the value of $\mathcal{E}$ the better the algorithm performs on the problem set L .

This measure compares not only the time complexity of the algorithms but it also evaluates their ability to solve the “difficult” inputs.

Chapter 4

Stochastic Search Algorithm for the Whitehead Minimization Problem

4.1 Introduction

At this point we would like to look at the Whitehead Reduction problem as a search problem. It was mentioned earlier that DWR is essentially a greedy descent algorithm, where on each iteration we reduce the length of the original word by solving the Length Reduction Problem. Due to the result of Whitehead (Theorem 2.1.3) this procedure will always find an optimal solution. The drawback is that the size of the neighborhood that needs to be searched on each iteration is proportional to the number of Whitehead automorphisms in the set Ω_n , which is exponential in rank of a free group. This makes the procedure inefficient for large ranks.

In this chapter we introduce a stochastic approach which combines the descent procedure with a heuristic search method. We use a genetic algorithm to solve the Length Reduction Problem on each iteration of the descent procedure. Due to the stochastic nature of the genetic algorithms we can use a smaller set of elementary

transformations which, we hope, will allow us to solve LR problem faster than the brute force enumeration. Indeed, our experiments show that the stochastic approach works faster on average in free groups with large enough ranks. Unfortunately, the new algorithm does not guarantee that it will hit the global optima and, therefore, the termination condition is not precisely defined. In Section 4.2.3 we address this problem.

In the following section we give the description of the stochastic algorithm and compute estimates for the running time on an input of given size. In Section 4.3 we experimentally justify the accuracy of the heuristic termination condition and compare the performance of stochastic algorithm with the standard deterministic algorithm DWR.

4.2 Description of the Algorithm

In this section we describe a Stochastic Search Algorithm for solving Whitehead's Reduction Problem (SWR). The general descriptions of stochastic search methods were given in Section 1.5. Below we give the detailed description of the major components of SWR: solution representation, solution evaluation, stochastic operators and termination conditions. The algorithm itself is described in Section 4.2.6.

4.2.1 Solutions and members of the population

Solutions to the Reduction Problem (Problem 2.1.4) are finite sequences of Whitehead automorphisms which carry a given tuple $U \in F^k$ to a minimal tuple U_{min} . As we have mentioned above one may use only automorphisms of the type 2 for this problem.

Moreover, not all automorphisms of the type 2 are needed as well (recall that a big number of such automorphisms is the main obstacle for the standard Whitehead algorithm DWR). What are optimal sets of automorphisms is an interesting problem. Our experiments show that the following set gives acceptable results.

Let $X = \{x_1, \dots, x_n\}$ and $F = F_n(X)$. Denote by $T = T_n$ the following set of Whitehead automorphisms:

$$\begin{aligned} W1: & \quad x_i \rightarrow x_i^{-1}, x_l \rightarrow x_l, l \neq i, i, l = 1, \dots, n, \\ W2: & \quad x_i \rightarrow x_j^{\pm 1} x_i, x_l \rightarrow x_l, j \neq i, l \neq i, i, j, l = 1, \dots, n, \\ W3: & \quad x_i \rightarrow x_i x_j^{\pm 1}, x_l \rightarrow x_l, j \neq i, l \neq i, i, j, l = 1, \dots, n, \\ W4: & \quad x_i \rightarrow x_j^{-1} x_i x_j, x_l \rightarrow x_l, j \neq i, l \neq i, i, j, l = 1, \dots, n. \end{aligned}$$

We call T the *restricted set* of Whitehead transformations. It is easy to see that the set of Nielsen automorphisms N_n is contained in T and, therefore, it follows from [49] that T generates $\text{Aut}(F)$. Hence any solution to Reduction Problem can be represented by a finite sequence of transformations from T . Notice that T has much fewer elements than Ω_n :

$$|T| = 5n^2 - 4n.$$

We define the search space $\mathcal{S}$ as the set of all finite sequences $\mu = \langle t_1, \dots, t_s \rangle$ of transformations from T . For such μ and a tuple $U \in F^k$ we define $U\mu = Ut_1 \dots t_s$.

At the beginning the algorithm generates an initial population by selecting N empty sequences. How to choose the size N of the initial (and all other) population is a non-trivial matter. It is clear that the greater the size N the larger the search space that is explored in one generation. But the trade off is that we may be spending too much time evaluating fitness value of members of the population. We used populations

with 50 members which seem to give satisfactory results.

4.2.2 Evaluation method

The *Fitness function* Fit provides a mechanism to assess members of a given population P .

Recall that the aim of SWR is to find a sequence of transformations $\mu = \langle t_1, \dots, t_s \rangle$, $t_i \in T$, such that

$$U\mu = U_{min}$$

for a given input $U \in F^k$. So members μ of a given population P with smaller total length $|U\mu|$ are closer to a solution, i.e., "fitter", than the other members. Therefore we define the fitness function Fit as

$$Fit(\mu) = \max_{\lambda \in P} \{|U\lambda|\} - |U\mu|.$$

Observe, that members with higher fitness values are closer to a solution U_{min} with respect to the metric on the graph $\Gamma(F, k, T)$. In fact, we have two different implementations of the evaluation criterion: the one as above, and another one in which a word is considered as a *cyclic* word, so we evaluate fitness values of cyclic permutations of $U\lambda$.

4.2.3 Termination condition

The *Termination condition* is a tool to check whether a given population contains a solution to the problem or not.

In the case of Whitehead method there are several ways to define a termination condition.

•

- T1) Once a new population P_n has been defined and all members of it have been evaluated one may check whether or not P_n contains a solution to Problem RP. To this end one can run "Elementary Length Reduction" routine (Procedure 2.2.1) on $U\mu^*$ for each fittest member $\mu^* \in P_n$ until U_{min} is found. Theoretically, it is a good termination condition, but, as we have mentioned already, to run ELR might be very costly.
- T2) If for a given tuple U we know in advance the length of a minimal tuple $|U_{min}|$, we define another (fast) termination condition as $|U\mu^*| = |U_{min}|$ for some fittest member $\mu^* \in P_n$. Such termination condition can be used, for example, when U is a part of a basis of F . All elements in U are primitive and the total length of U_{min} is simply equal to the number of elements in U .
- T3) Suppose now that we do not know $|U_{min}|$ in advance, but we know the expected number of populations, say $E = E(U)$ (or some estimates for it) which is required for the genetic algorithm SWR to find U_{min} when starting on a tuple U . In this case we can use the following strategy: if the algorithm keeps working without improving on the fitness value $Fit(\mu^*)$ of the fittest members μ^* for long enough, say for the last pE generations (where $p \geq 1$ is a fixed constant), then it halts and gives $U\mu^*$ for some fittest μ^* as an outcome.

If the number $E = E(U)$ is sufficiently small this termination condition could be efficient enough. In Section 4.3.2 we will describe some techniques and numerical results on how one can estimate the number $E(U)$. Of course, in this case there is no guarantee that the tuple $U\mu^*$ is indeed minimal. We refer to such termination conditions as to *heuristic* ones, while the condition T1 is

deterministic.

T4) In Chapter 5 we describe a Classification System which with a very high accuracy decides whether a given element is Whitehead minimal or not. The classifier can be used to define another heuristic termination condition. Let $decide(w) \in \{0, 1\}$ be the decision of the classifier for a given word w , where 1 means w is minimal and 0 it is not. We terminate the algorithm if for some fittest member $\mu^* \in P$

$$decide(w) = 1, \forall w \in U\mu^*.$$

T5) One can combine conditions T3(T4) and T1 in the following way. The algorithm uses the heuristic termination conditions and then checks (using T1) whether or not the output $U\mu^*$ is indeed minimal. It is less costly then T1 (since we do not apply T1 at every generation) and it is more costly then T3 or T4.

4.2.4 Stochastic operators

There are five basic random operators that were used in the algorithm.

One point crossover

Let $\mu_1 = \langle t_1, \dots, t_e \rangle$ and $\mu_2 = \langle s_1, \dots, s_l \rangle$ be two members of a population P_n which are chosen with respect to some selection method. Given two random numbers $0 < p < e$ and $0 < q < l$ the algorithm constructs two offspring o_1 and o_2 by *recombination* as follows:

$$o_1 = \langle t_1, \dots, t_{p-1}, s_q, \dots, s_l \rangle, \quad o_2 = \langle s_1, \dots, s_{q-1}, t_p, \dots, t_e \rangle.$$

Mutations

The other four operators M_{att} , M_{ins} , M_{del} , M_{rep} act on a single member of a population and are usually called *mutations*. They attach, insert, delete, or replace some transformation in a member. Namely, let $\mu = \langle t_1, \dots, t_l \rangle$ be a member of a population. Then:

M_{att} *attaches* a random transformation $s \in T$

$$M_{att} : \langle t_1, \dots, t_l \rangle \rightarrow \langle t_1, \dots, t_l, s \rangle;$$

M_{ins} *inserts* a random transformation $s \in T$ into a randomly chosen position i

$$M_{ins} : \langle t_1, \dots, t_l \rangle \rightarrow \langle t_1, \dots, t_{i-1}, s, t_i, \dots, t_l \rangle;$$

M_{del} *deletes* the transformation in a randomly chosen position i

$$M_{del} : \langle t_1, \dots, t_l \rangle \rightarrow \langle t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_l \rangle;$$

M_{rep} *replaces* the randomly chosen t_i by a randomly chosen $s \in T$

$$M_{rep} : \langle t_1, \dots, t_l \rangle \rightarrow \langle t_1, \dots, t_{i-1}, s, t_{i+1}, \dots, t_l \rangle .$$

Operator M_{att} is a special case of M_{ins} , but it is convenient to have it as separate operator (see remarks in the Section 4.2.5).

Replacement

In this section we discuss a protocol to construct members of the next population P_{new} from the current population P .

First, we select randomly two members μ, λ from P . The probability to choose a member from P is equal to

$$Pr(m) = \frac{Fit(m)}{\sum_{m_i \in P} Fit(m_i)}.$$

With small probability (0.10 - 0.15) we add both μ and λ to an intermediate population P'_{new} . Otherwise, we apply the crossover operator to μ and λ and add the offspring to P'_{new} . We repeat this step until we get the required number of members in P'_{new} (in our case 50).

Secondly, to every member $m \in P'_{new}$ we apply a random mutation M with probability 0.85 and add the altered member to the new population P_{new} . The choice of M is governed by the corresponding probabilities p_M . Otherwise (with probability 0.15) we add the member m to P_{new} unchanged. We refer to Section 4.2.5 for a discussion of our choice of the probabilities p_M .

In addition the solution with the highest fitness value among all previously occurred solutions is always added to the new population (replacing a weakest one). This implies that if we denote by μ_n one of the fittest members of a population P_n then

$$|U\mu_0| \geq |U\mu_1| \geq \dots$$

4.2.5 Precise solutions and local search

It has been shown that different heuristics and randomized methods can be combined together, often resulting in more efficient *hybrid* algorithms. Genetic algorithms are good in covering large areas of the search space. However, they may fail when a more thorough trace of a local neighborhood is required. In case of symbolic computations this becomes an important issue since we are looking for an exact solution, not an

approximate one. Even if the current best member of a population is one step away from the optimum it might take some time for the standard genetic algorithm to find it. In our case, experiments show that the standard genetic algorithms can quickly reach the neighborhood of the optimum, but it may be stuck being unable to hit the right solution. To avoid that one could add a variation of the local search procedures to the standard genetic algorithm.

In SWR a kind of gradient descent procedure was implicitly introduced via mutation operators. Observe, that in general, if $M \neq M_{att}$ then for a given member μ the tuple $UM(\mu)$ lies far apart from $U\mu$ in the graph $\Gamma(F, k, T)$. However, the mutation M_{att} always gives a tuple $UM_{att}(\mu)$ at distance 1 from $U\mu$ in the graph $\Gamma(F, k, T)$. Therefore, the greater chance to apply M_{att} , the more neighbors of $U\mu$ we can explore. It was shown experimentally that SWR performs much better when M_{att} has a greater chance to occur. We used $p_{M_{att}} = 0.7$, and $p_M = 0.1$ for $M \neq M_{att}$.

4.2.6 Steps of SWR

The algorithm is executed in the sequence described in Figure 4.1. First, we initialize population P_1 with randomly generated sequences of transformations of length 1 and compute the corresponding fitness values. On each iteration of the descent procedure we repeat steps 4-7. On the step 5 we construct new population by applying random operators as discussed in Section 4.2.4. If a shorter solution is found we perform the descent step and implicitly restart the genetic search procedure for the new shortest image $U_0\mu_{max}$ of the input tuple U_0 (steps 8,9). We continue to iterate until the termination condition is met.

Operation $\circ$ (step 9) is a regular concatenation of the sequences μ_{max} and μ^*

```

PROCEDURE SWR ( $U_0$ )
1:      Randomly Initialize  $P_1$ 
2:      SET  $U' = U_0$ 
3:      SET  $\mu_{max}$ , s.t.  $Fit(U', \mu_{max}) = \max\{Fit(U', \mu) \mid \forall \mu \in P_1\}$ 
4:      WHILE NOT termination condition  $TC(U_0\mu_{max})$ 
5:          SET  $P_{n+1} = Replacement(Mutation(Crossover(P_n)))$ 
6:          SET  $\mu^* = \max\{Fit(U', \mu) \mid \forall \mu \in P_{n+1}\}$ 
7:          IF  $Fit(U', \mu^*) > Fit(U_0, \mu_{max})$ 
8:              SET  $\mu_{max} = \mu_{max} \circ \mu^*$ 
9:              SET  $U' = U_0\mu_{max}$ 
10:         Randomly Initialize  $P_{n+1}$ 
11:         END IF
12:     END WHILE

```

Figure 4.1: Algorithm SWR.

which implements the composition of the corresponding automorphisms. Notice that at every stage of the algorithm, $U_0\mu_{max}$ is the shortest tuple in $Orb(U_0)$ found so far. When SWR terminates, sequence μ_{max} contains a step by step proof that the shortest tuple U' is, indeed, in $Orb(U_0)$.

To find a shorter solution, one might reduce the length of μ_{max} by deleting all subsequences in μ_{max} which introduce loops in the automorphisms graph Γ . Let $\mu_{max} = \langle t_1, \dots, t_l \rangle$. We remove all subsequences $\langle t_i, \dots, t_{i+k} \rangle$ of μ_{max} such that

$$U_0 t_1 \dots t_{i-1} = U_0 t_1 \dots t_{i+k}.$$

4.3 Complexity Estimates and Performance evaluation

It is not easy to estimate, or even to define, time complexity of SWR because of its stochastic nature. However, one can estimate the time complexity of the major components of SWR on each given iteration. Afterward, one may define a time complexity function $T_{SWR}(s)$ as an average number of iterations required by SWR to find a solution starting on a given input s .

In the following section we give experimental results showing that the number of iterations of the algorithm SWR required to solve the Length Reduction Problem does not depend on the length of an input word. In Section 4.3.2 we estimate the expected running time of SWR and give a conservative bound for the termination condition T3. In the concluding sections we experimentally evaluate the accuracy of the termination condition T3 and compare SWR with the original algorithm DWR.

4.3.1 Complexity of Length Reduction Problem

In this section we test the genetic search algorithm on the length reduction problem LRP, which is the main component of the Whitehead's Method.

To this end we generate a finite set $S_{WC1}(r)$ of non-minimal elements in a free group F_r , for $r = 2, 5, 10, 15, 20$, by applying random Whitehead automorphisms to elements from S_F as was described in Section 3.2.3. Notice that elements in $S_{WC1}(r)$ are not randomly chosen non-minimal elements from F , they are non-minimal elements at distance 1 from minimal ones.

The results of our experiments indicate that the average time required for SWR

to find a length reducing Whitehead automorphism for a given non-minimal element $w \in S_{WC1}(r)$ does not depend significantly on the length of the word w .

Let $T_{gen}(w)$ be the number of iterations required for SWR to find a length-reducing automorphism for a given $w \in F$ during a particular run of SWR on the input w . We compute the average value of $T_{gen}(w)$ on inputs $w \in S_{WC1}(r)$ of a given "size". If the length of a word w is taken as its size then we obtain the following time complexity function with respect to the test data $S_{WC1}(r)$:

$$T_r(m) = \frac{1}{|S_m|} \sum_{w \in S_m} T_{gen}(w)$$

where $S_m = \{w \in S_{WC1}(r) \mid |w| = m\}$.

Values of $T_r(m)$ are presented in Figure 4.2 for free groups F_r with $r = 2, 5, 10, 15, 20$.

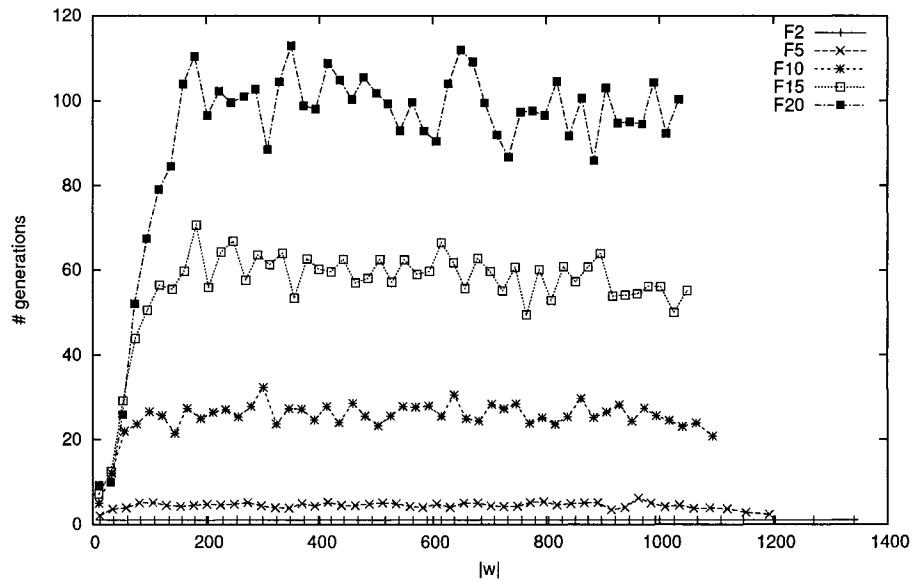


Figure 4.2: Values of T_r , $r = 2, 5, 10, 15, 20$. T_r stabilizes as length of the input elements grows.

We can see from the graphs that the function T_r grows for small values of $|w|$ and then stabilizes at some constant value T_r^* . It seems that T_r does not significantly

depend on the word's length and depends only on the rank r (for long enough words w).

In Table 4.1 we give correlation coefficients between T_r and $|w|$ for $r = 2, 5, 10, 15, 20$, which are sufficiently small. Moreover, they are negative for large words, which means that T_r does not increase when the word's length increases.

	F_2	F_5	F_{10}	F_{15}	F_{20}
all words	-0.012	-0.016	0.015	0.03	0.072
$ w > 100$	-0.011	-0.03	-0.019	-0.025	-0.005

Table 4.1: Correlation between $|w|$ and T_r , $r = 2, 5, 10, 15, 20$. Correlation is negative for large words which means that the number of generations T_r does not increase with the input size.

We summarize the discussion above in the following statements.

Conclusion 4.3.1. *The upper bound on the number of iterations required for SWR to find a length reducing automorphism for a given non-minimal input w does not depend on the length of $|w|$, it depends only on the rank r (for long enough input words).*

Recall that a similar phenomena was observed for the deterministic Whitehead's algorithm in Conclusion 2.3.3.

Conclusion 4.3.2. *One has to replace the length size function by Whitehead Complexity function when measuring the time-complexity of the Length Reduction Problem.*

Conclusion 4.3.3. *For each free group F_r the time-complexity function T_r is bounded from above by some constant value T_r^* .*

We can try to estimate the value T_r^* as the *expected number of generations*

$$E(r) = \frac{1}{|S_{WC1}(r)|} \sum_{w \in S_{WC1}(r)} T_{gen}(w).$$

required for SWR to find a length-reducing automorphism for generic non-minimal elements from F_r .

Of course, the conclusions above are not mathematical theorems, they are just empirical phenomena that can be seen from our experiments based on the test set $S_{WC1}(r)$.

It is important to make sure that the set $S_{WC1}(r)$ is sufficiently representative. To this end, we made sure, firstly, that the distributions of lengths of words from the set $S_{WC1}(r)$ are similar for different ranks (using the variable l). Secondly, our choice of the parameter K in the construction of $S_{WC1}(r)$ ensures representativeness of the test data with respect to the characteristic $E(r)$. Namely, we want to make sure that values of T_r^* are not significantly different from the estimate $E(r)$. We use method described in Section 3.3.2 to choose K such that we are guaranteed that $|T_r^* - E(r)| \leq .5$ or that $E(r)$ approximates T_r^* within one generation. The actual values of $E(r)$ and the corresponding confidence are given in Table 4.2.

	F_2	F_5	F_{10}	F_{15}	F_{20}
$E(r)$	1.0	2.3	6.5	11.3	17.7
$Prob(T_r^* - E(r) < .5)$	1.00	1.00	1.00	.99	.99

Table 4.2: Values of $E(r)$ and the confidence for $r = 2, 5, 10, 15, 20$.

Remark 4.3.1. Notice that the set $S_{WC1}(r)$ is a pseudo-random sampling of elements $w \in F_r$ with $WC(w) = 1$. This explains the behavior of the function T_r in Figure 4.2.

The number of iterations required for SWR to find a length reducing automorphism depends on Whitehead complexity not on the lengths of the words.

For the stochastic algorithm SWR one can define an average time complexity function $T_{r,Y}(m)$ with respect to the test data $S_{WC_1}(r)$ and the "size" function WC_Y as follows:

$$T_{r,Y}(m) = \frac{1}{|S_m|} \sum_{w \in S_m} T_{gen}(w)$$

where $S_m = \{w \in S_{WC_1} \mid WC_Y(w) = m\}$.

Conjecture 4.3.4. *The average number of iterations required for SWR to find w_{min} on an input $w \in F$ depends only on $WC(w)$ and the rank of the group F .*

We discuss some experiments made to verify Conjecture 4.3.4 in Section 4.3.3.

4.3.2 Complexity Estimates of Components of SWR

Let SWR be executed on an input $w \in F$. Below we give some estimates for the time required for SWR to make one iteration. It is easy to see that the total execution time $T_{CMR}(P)$ of Crossover, Mutation, and Replacement operators, needed to generate the population P_{new} from a given population P , does not depend on the length of the input w and depends only on the cardinality of the population P (which is fixed), and the length $|\mu|$ of members μ of the current population P (here $|\mu|$ is the length of the sequence μ). Therefore, for some constant C_{CMR} the following estimate holds

$$T_{CMR}(P) \leq C_{CMR} \cdot M_P$$

where $M_P = \max\{|\mu| \mid \mu \in P\}$.

To compute $Fit(\mu)$ for a given $\mu \in P$ requires running the substitution routine SR (Procedure 2.3.2) on the input $w\mu$. Since $|wt| \leq 3|w|$ for any restricted Whitehead automorphism $t \in T$ one has $|w\mu| \leq 3^{|\mu|}|w|$ for each $\mu \in P$. Hence the execution time T_{Fit} required to compute $Fit(\mu)$ can be bounded from above by

$$T_{Fit} \leq C_{Fit} \cdot |w\mu| \leq C_{Fit} \cdot 3^{M_P} \cdot |w|$$

This argument shows that the time $T_{gen}(P)$ required for SWR to generate a new population from a given P can be estimated from above by

$$T_{gen}(P) \leq T_{CMR}(P) + T_{Fit} \leq C_{CMR} \cdot M_P + C_{Fit} \cdot 3^{M_P} \cdot |w|.$$

In fact, the estimate $|wt| \leq 3|w|$ is very crude, as we have seen in Section 2.5 one has on average $|wt| \leq c_r|w|$ and the values of c_r are much smaller than 3 (see Table 2.2). So on average one can make the following estimate:

$$T_{gen}(P) \leq C_{CMR} \cdot M_P + C_{Fit} \cdot c_r^{M_P} \cdot |w|.$$

Thus, the length of members of the current population P has crucial impact on the time complexity of the procedure that generates the next population.

A priori, there are no limits on the length of the population members $\mu \in P$. However, the descent procedure divides SWR into a sequence of separate runs, each of which solves the Length Reduction Problem for a current word $w_i = wt_1 \dots t_i$. Furthermore, our experiments show that to solve this problem SWR generates population members in P of the average length which *does not depend* on the length of the input w_i , it depends only on the rank of F . In Figure 4.3 we present results of our experiments with computing $|\mu|$, ($\mu \in P$) when running SWR on inputs w from $S_{WC1}(r)$.

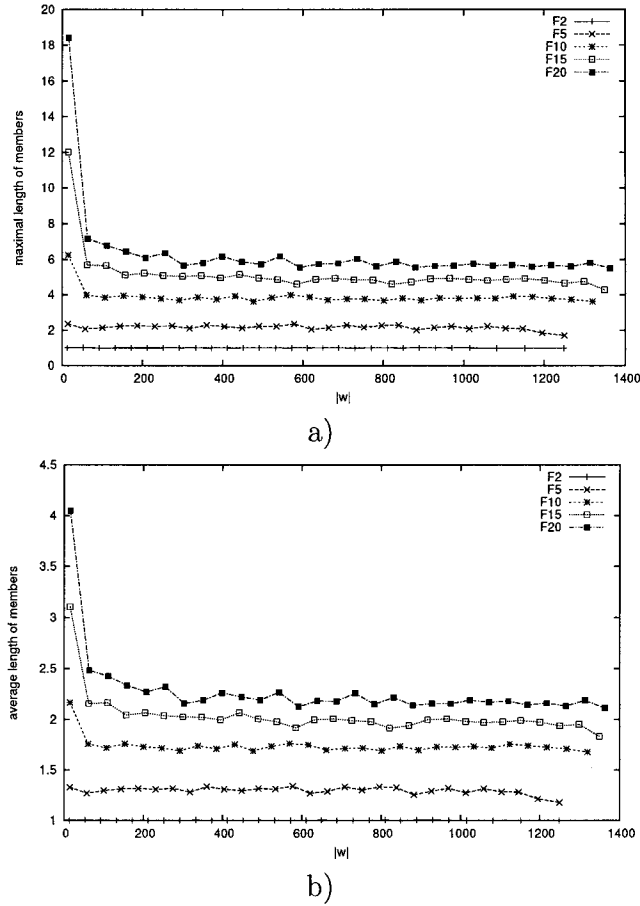


Figure 4.3: Values of $|\mu|$ for various word lengths: a) maximal $|\mu|$, b) average $|\mu|$. For most inputs $|\mu|$ can be bounded by a constant.

In Table 4.3 we collect average and maximal values of $|\mu|$ for inputs $w \in S_{WC1}(r)$ for various ranks r .

This experimental data allows us to state the following observed phenomena.

Conclusion 4.3.5. *To solve the Length Reduction problem for a given non-minimal $w \in F$ SWR generates new populations in time bounded from above by $C_r|w|$ where C_r is a constant bounded from above in the worst case by*

$$C_r \leq C_{CMR} \cdot M_P + C_{Fit} \cdot 3^{M_P},$$

	F_2	F_5	F_{10}	F_{15}	F_{20}
Average $ \mu $	1.0	1.3	1.7	2.0	2.3
Maximal $ \mu $	1.0	2.2	3.8	5.1	6.3

Table 4.3: Maximal and average lengths of the population members.

and on average by

$$C_r \leq C_{CMR} \cdot M_P + C_{Fit} \cdot c_r^{M_P}.$$

Now we can estimate the expected time-complexity $T_{SWR_r(w)}$ of SWR on an input $w \in F_r$ as follows:

$$T_{SWR_r(w)} \approx Gen_r(w) \cdot average(T_{gen}(P)) \leq E(r) \cdot WC_T(w) \cdot C_r \cdot |w|.$$

4.3.3 Experiments with primitive elements

In this section we discuss results of experiments with primitive elements. Experimenting with primitive elements has several important advantages:

- in general, primitive elements w require long chains of Whitehead automorphisms (relative to $|w|$) to get to w_{min} ,
- one can easily generate pseudo-random primitive elements,
- the genetic algorithm SWR has a perfect termination condition $|w_{min}| = 1$ for primitive elements w .

Thus, primitive elements provide an optimal test data to compare various modifications of Whitehead algorithm and to verify (experimentally) the conjectures and conclusions stated in the previous sections.

Again, for each free group F_r ($r = 2, 5, 10, 15, 20$), we construct a set $S_P(r)$ of test primitive elements.

We use the data sets $S_P(r)$ to verify, using independent experiments, the conclusions of Section 4.3.1 on the average expected time $E(r)$ required for SWR to solve the length reduction problem in the group F_r . If they are true then the expected number of iterations $Gen_r(w)$ required for SWR to produce w_{min} for a given input $w \in F_r$ satisfies the following estimate:

$$Gen_r(w) \leq cE(r)WC(w) \leq cE(r)|w| \quad (4.3.1)$$

Let $Q_{r,c}$ be the fraction of such elements w in the set $S_P(r)$ for which $Gen_r(w) \leq cE(r)|w|$ holds. Table 4.4 shows values of $Q_{r,c}$ for $r = 2, 5, 10, 15, 20$ and $c = 1$. We can see that $Q_{r,1}$ is close to 1 for all tested ranks, as predicted.

In particular, we can make the following

Conclusion 4.3.6. *The genetic algorithm SWR with the termination condition T3 gives reliable results even with the constant $c = 1$.*

	F_2	F_5	F_{10}	F_{15}	F_{20}
$E(r)$	1	3	7	12	18
all words	.93	.93	.99	.99	.99
$ w > 100$	1.0	.99	.99	.99	1.0

Table 4.4: Fraction of elements $Q_{r,1}$, $r = 2, 5, 10, 15, 20$ for which $Gen_r(w) \geq E(r)|w|$, where $E(r)$ is the average expected number of generations required for SWR to solve the length reduction problem. The termination condition T3 gives reliable results even with $c = 1$.

In constructing the set $S_P(r)$ we select K to ensure the representativeness of characteristic $Q_{r,1}$. We chose K such that probability

$$\text{Prob}(|Q_{r,1} - p| < \epsilon) \approx 0.95,$$

where p is a probability that inequality 4.3.1 holds and ϵ is small enough (less than 0.01). The approximate 95% confidence intervals $[P_L, P_U]$ for sets $S_P(r)$ generated with $K = 500$ are given in Table 4.5.

	F_2	F_5	F_{10}	F_{15}	F_{20}
$Q_{r,1}$	.930	.927	.996	.995	.992
$[P_L, P_U]$	[.925,.935]	[.922,.932]	[.995,.997]	[.994,.996]	[.991,.993]
$[P_L - Q_{r,1}, P_U - Q_{r,1}]$	[-.005,.005]	[-.005,.005]	[-.001,.001]	[-.001,.001]	[-.001,.001]

Table 4.5: Values of $Q_{r,1}$ and the corresponding 95% confidence intervals.

4.3.4 Comparison of the Standard Whitehead algorithm DWR with the Stochastic Whitehead algorithm SWR

In this section we compare results of our experiments with the standard Whitehead algorithm DWR and the stochastic algorithm SWR. We tested these algorithms on the set S_P of pseudo-random primitive elements.

As we have seen in Section 4.3.2 we may estimate the expected time required for SWR to find a length reducing automorphism on a non-minimal input $w \in F_r$ as:

$$C_r \cdot E(r) \cdot |w|.$$

Recall from Section 2.3.1 that the expected time required for DWR to find such an

automorphism can be estimated by

$$\frac{A_r}{|LR_r|} \cdot |w|.$$

In Table 4.2 and Figure 2.2 we collected an experimental data on average values of $E(r)$ and $\frac{A_r}{|LR_r|}$ for various free groups F_r . It seems from our experiments that

$$C_r \cdot E(r) \ll \frac{A_r}{|LR_r|}$$

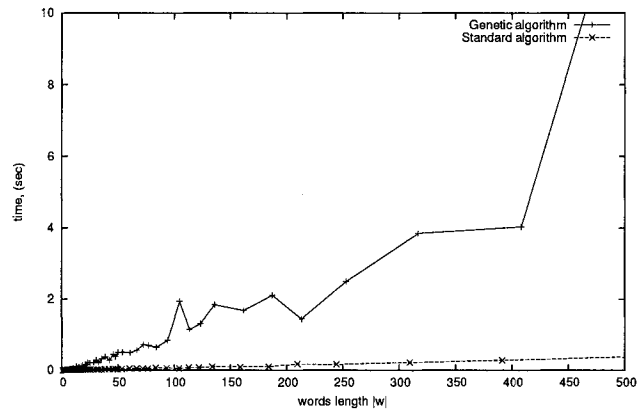
for big enough r . Thus, we should expect much better performance of SWR than DWR on groups of higher ranks.

In Table 4.6 and Figures 4.4 we present results on performance comparison of SWR with an implementation of the standard Whitehead's algorithm DWR available in *MAGNUS* software package [5]. We run the algorithms on words $w \in S_P(r)$ and measure the execution time. We terminated an algorithm if it was unable to obtain the minimal element (of length 1) on an input w after being running for more then an hour. There were very few runs of DWR for words $w \in F_{10}$ with $|w| > 100$ that finished within an hour. There were no such runs for $|w| > 200$ at all, and therefore results of these experiments are marked "na" (not available). For the reason of DWR running very slow, comparisons for the groups of ranks larger then 10 have not been done at all.

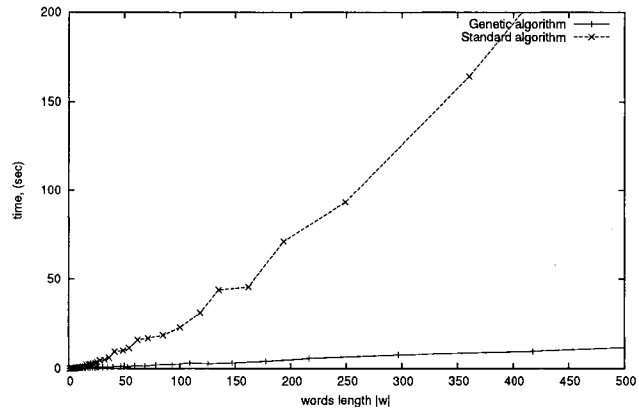
Conclusion 4.3.7. *SWR performs much better than DWR in free groups F_r for sufficiently big r (in our experiments, $r \geq 5$) and on sufficiently long inputs (in our experiments, $|w| \geq 10$).*

	F_2			F_5			F_{10}		
$ U $	57	104	268	57	106	228	52	102	268
Time spent by the standard algorithm, s	.03	.07	.18	13.29	27.4	85.9	1995	na	na
Time spent by the genetic algorithm, s	.52	1.2	2.7	1.4	2.6	5.6	2.6	6.07	17.4

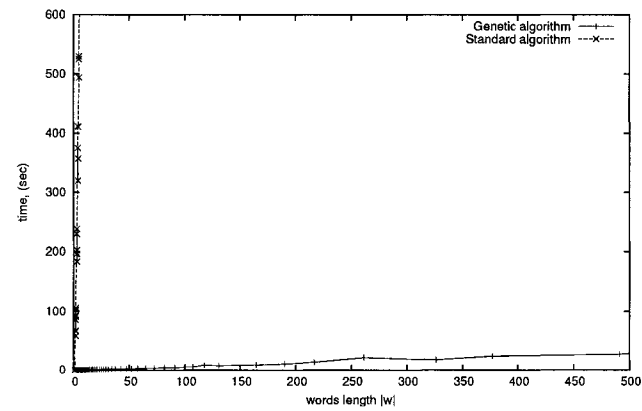
Table 4.6: Performance comparison of DWR and SWR. SWR significantly outperforms DWR in free groups with large ranks.



a)



b)



c)

Figure 4.4: Time comparison between standard and genetic algorithms on primitive elements in a) F_2 , b) F_5 and c) F_{10} .

Chapter 5

Recognition of Whitehead-minimal elements

5.1 Introduction

In the previous chapter we showed that the reduction problem can be solved faster on average by introducing a stochastic component into the search procedure. Note, that if the input element is non-minimal, the stochastic procedure will eventually reduce it and return the corresponding solution. However, it does not provide an efficient way of deciding whether the element is already minimal.

In this chapter we introduce a pattern classification system WMIN that recognizes Whitehead-minimal elements in a given free group. The corresponding probabilistic classification algorithm is very fast (real time algorithm) and recognizes minimal words correctly with very high probability. There are two modifications of the system based on linear regression and Support Vector Machines. The latter approach is more suitable for the groups with large ranks.

The classification system does not use any particular results from group theory. To the contrary, the analysis of the results of the classifications led to the discovery of

some properties of non-minimal elements in the free group of rank 2. We have found a very simple pattern recognition system that partitions all non-minimal elements in F_2 into two clusters M_1 and M_2 and also it partitions the set of all elementary Whitehead automorphisms into two subsets T_1 and T_2 such that with high probability only automorphisms from T_i can reduce the length of elements in $M_i, i = 1, 2$. This allows one to reduce the search for a length reducing automorphism for a given $w \in F_2$ by half. We return to this phenomena in the next chapter where we use cluster analysis of the non-minimal elements to develop heuristics for solving the Length Reduction Problem.

This chapter is organized as follows. In Section 5.2 we give the description of the system itself. In Section 5.3 we describe experiments with elements in the free group of rank 2. In Section 5.4 we give results of experiments with free groups F_3, F_4 and F_5 and discuss the limitations of the regression approach. Section 5.5 introduces a classification system based on Support Vector Machines and presents the corresponding results for groups with large ranks.

5.2 Description of the system

WMIN is based on a supervised learning approach. Development of any supervised classifier consists of defining the following components:

1. Data acquisition/generation procedure.
2. Feature representation.
3. The assumed underlying model and the decision rule.

4. The evaluation criterion.

In this section we describe the components of the classification system WMIN.

5.2.1 Training Data Set

A classifier for minimal elements in a free group F has to determine for an arbitrary given element $w \in F$ whether w is minimal or not. In fact, it is convenient to consider the Reduction Problem only for cyclically reduced words in F .

Lemma 5.2.1. *Every minimal word w_{min} in F is cyclically reduced.*

Proof. First we show that

$$\tilde{w} \in Orb(w),$$

where $\tilde{w}$ is the cyclically reduced form of w . Observe, that

$$w = g^{-1}\tilde{w}g,$$

for some word $g \in F$. Showing that $\tilde{w} \in Orb(w)$ is equivalent to showing that $w \in Orb(\tilde{w})$, therefore we need to show that

$$g^{-1}\tilde{w}g \in Orb(\tilde{w}).$$

Consider the transformation

$$\varphi_g : x \rightarrow g^{-1}xg, \quad \forall x \in X$$

where $g \in F$ and x are the generators from X .

We claim that

$$\varphi_g(u) = g^{-1}ug,$$

for some word $u \in F$. The proof is by induction on the length of u . Obviously

$$\varphi_g(x) = g^{-1}xg, \quad x \in X^{\pm 1}.$$

Now let $u = u_1u_2$ and $|u_i| \geq 1$, then

$$\varphi_g(u) = \varphi_g(u_1u_2) = g^{-1}u_1gg^{-1}u_2g = g^{-1}u_1u_2g = g^{-1}ug.$$

Now we show that φ_g is an automorphism. In order to do that, we need to show that φ_g is a homomorphism and that it has the inverse. Obviously, φ_g is defined on all the elements from F and

$$\varphi_g(uv) = g^{-1}uv g = g^{-1}u(gg^{-1})vg = (g^{-1}ug)(g^{-1}vg) = \varphi_g(u)\varphi_g(v).$$

Therefore, φ_g is a homomorphism. The inverse of φ_g :

$$\varphi_g^{-1}(u) = gug^{-1}.$$

Indeed,

$$\varphi_g^{-1}(\varphi_g(u)) = \varphi_g^{-1}(g^{-1}ug) = gg^{-1}ugg^{-1} = u.$$

Therefore, φ_g is an automorphism. Observe, that

$$\varphi_g(\tilde{w}) = g^{-1}\tilde{w}g = w.$$

It follows that $\tilde{w} \in Orb(w)$.

Now, suppose w is a minimal element and there exists its cyclically reduced form $\tilde{w}$ such that

$$|\tilde{w}| < |w|.$$

But we have showed that $\tilde{w} \in Orb(w)$ and therefore it contradicts the minimality condition of w . This completes the proof. ■

There exists a "very fast" algorithm to produce a cyclically reduced form of a reduced word w .

Procedure 5.2.2. *Let $w = t_1 \dots t_l$ and w is reduced. To produce a cyclically reduced word, remove maximal subwords $u = t_1 \dots t_k$ and $v = t_{l-k} \dots t_l$ such that $u = v^{-1}$, where $k \leq (l-1)/2$.*

It suffices to construct a classifier for cyclically reduced words in F .

At first glance, the obvious choice for the training dataset would be the set of randomly generated cyclically reduced words from F . However, it has been argued in Section 3.2.1 that randomly taken cyclic words in F are already minimal with asymptotic probability 1. Therefore, a set of randomly generated words would be highly biased toward the class of minimal elements. To obtain fair numbers of representatives from both classes we use the following procedure to generate training datasets.

For each positive integer $l = 1, \dots, L$ we generate (pseudo-) randomly and uniformly K cyclically reduced words from $F(X)$ of length l , where $L = 1000$ and $K = 10$ were chosen for practical reasons.

Denote the resulting set by W . Then using the deterministic Whitehead algorithm one can effectively generate the corresponding set of minimal elements

$$W_{min} = \{w_{min} \mid w \in W\}.$$

Notice, that for a free group with large rank the generation of W_{min} might be time consuming. In such a case the stochastic algorithm SWR can be used.

With probability 0.5 we substitute each $v \in W_{min}$ with the cyclic word $\tilde{v}t$, where t is a randomly and uniformly chosen automorphism from $\Omega(X)$ such that $|\tilde{v}t| > |v|$

(if $|\widetilde{v}t| = |v|$ we chose another $t \in \Omega(X)$, and so on). Now, the resulting set L is a set of pseudo-randomly generated cyclically reduced words representing the classes of minimal and non-minimal elements in approximately equal proportions. However, it seems, the class of non-minimal elements is not quite representative, since every one its element w has Whitehead complexity 1, i.e., there exists a single Whitehead automorphism which reduces w to w_{min} (see Section 2.5 for details on Whitehead complexity). We will see in the following sections that the set described above is a sufficiently good training dataset which is much easier to generate than a set with uniformly distributed Whitehead complexity of elements.

From the construction of the set L we know for each element $v \in L$ whether or not it is minimal. Finally, we construct the training set

$$D = \{ \langle v, P(v) \rangle \mid v \in L \},$$

where

$$P(v) = \begin{cases} 1, & v \text{ is minimal;} \\ 0, & \text{otherwise.} \end{cases}$$

5.2.2 Features

Let w be a reduced word in the alphabet $\in X^{\pm 1}$. In this section we describe the features of w which characterize a certain placement of specific words from $F(X)$ in w .

First, denote by $C(w, v)$ the number of times word $v \in F(X)$ occurs in w as a subword.

Let $K \in \mathbb{N}$ be a natural number, $v_1, \dots, v_K \in F(X)$ be words from $F(X)$, and

$U_1, \dots, U_{K+1} \subseteq F(X)$ be subsets of $F(X)$. Denote by

$$C(w, U_1 v_1 \dots v_K U_{K+1})$$

the number of times words of the type

$$u_1 v_1 u_2 \dots v_K u_{K+1},$$

where $u_j \in U_j$, occur in w as subwords. For fixed $K, v_1, \dots, v_K, U_1, \dots, U_{K+1}$ we obtain a *counting function*

$$w \in F \longrightarrow C(w, U_1 v_1 \dots v_K U_{K+1}) \in \mathbb{N}_+ \quad (5.2.1)$$

The normalized value

$$\frac{1}{|w|} C(w, U_1 v_1 \dots v_K U_{K+1})$$

is called a *feature* of w and the function

$$w \in F \longrightarrow \frac{1}{|w|} C(w, U_1 v_1 \dots v_K U_{K+1}) \in \mathbb{R}$$

is called a *feature function* on F . Usually we omit U_i in our notations if $U_i = \emptyset$.

If $\bar{C} = (C_1(w), \dots, C_N(w))$ is a sequence of counting functions like (5.2.1) one can associate with w a vector of real numbers:

$$f_{\bar{C}}(w) = \frac{1}{|w|} \langle C_1(w), \dots, C_N(w) \rangle \in \mathbb{R}^N$$

which is called a *feature vector* of w . Every choice of the sequence $\bar{C}$ gives a vector $f_{\bar{C}}(w)$ which reflects the structure of w .

For example, if $a \in X^{\pm 1}$ then $C(w, a)$ counts the number of occurrences of the letter a in w . The feature vector (where for simplicity we assume that the components are written in some order which we do not specify)

$$f_0(w) = \frac{1}{|w|} \langle C(w, a) \mid a \in X^{\pm 1} \rangle$$

shows the frequencies of occurrences of letters from $X^{\pm 1}$ in w .

Let $V_n = \{v \mid v \in F, |v| = n\}$ and $W_n = \bigcup_{i=0}^n V_i$. The feature vector

$$f_1(w) = \frac{1}{|w|} \langle C(w, v) \mid v \in V_2 \rangle$$

shows the numbers of occurrences of words of length two in w relative to the length of w . If $x_1, x_2 \in X^{\pm 1}$ then the counting function $C(x_1 V_1 x_2)$ gives the number of occurrences of x_1 and x_2 in w one letter apart.

To visualize some structures described by the counting functions above we associate with a given word $w \in F(X)$ a weighted labelled directed graph $\Gamma(w)$. Put $V(\Gamma(w)) = X^{\pm 1}$. For given $x, y \in X^{\pm 1}$ and $v \in F(X)$ we connect the vertex x to the vertex y by an edge with a label v and weight $C(w, xvy)$. Now, with every edge from x to y with label xvy one can associate a counting function $C(w, xvy)$, and vice versa. It follows that every subgraph Γ of $\Gamma(w)$ gives rise to a particular set of counting functions $\bar{C}_\Gamma$ of the type $C(w, xvy)$, and conversely, every set $\bar{C}$ of counting functions of the type $C(w, xvy)$ determines a subgraph $\Gamma_{\bar{C}}$ of $\Gamma(w)$.

For instance, the feature mapping f_1 corresponds to the subgraph $\Gamma_1(w)$ of $\Gamma(w)$ which is in a sense a directed version of the Whitehead Graph $WG(w)$ (Definition 3.4.1) of w .

In addition we define a feature mapping based directly on $WG(w)$:

$$f_{WG}(w) = \langle l_1, \dots, l_k \rangle,$$

where l_i 's are the weights of the Whitehead Graph $WG(w)$ as defined in Section 3.4.3. Notice that mapping $f_{WG}(w)$ has two times less features than the corresponding feature mapping $f_1(w)$.

5.2.3 Model

The classification algorithm has to predict the value $P(w)$ of the predicate P for a given word w . One of the approaches is to explore the relationship between $P(w)$ and the corresponding feature vector $v = f(w)$ of the word w . We can try to approximate the value of $P(w)$ via a linear function on $f(w)$:

$$P(w) \approx \beta' f(w),$$

where β is an unknown column vector of coefficients. Inferring β from the training set is the task of the classical linear regression analysis (see Section 1.3.2). Given a dataset

$$D = \{(w_i, P(w_i)) \mid i = 1, \dots, n\}$$

one can compute the feature vectors $f(w_i), i = 1, \dots, n$, and form the standard regression model as

$$\mathcal{P} = V\beta + \epsilon,$$

where $\mathcal{P} = \langle P(w_1), \dots, P(w_n) \rangle$ is a vector (column) of the known values of P , V is the matrix

$$V = [f(w_1), \dots, f(w_n)]$$

with the feature vectors as rows, β is a vector of unknown regression coefficients, and ϵ represents the approximation error. Using the least squares method we find β such that the mean square error

$$\|\mathcal{P} - V\beta\|^2 = \|\epsilon\|^2$$

is as minimal as possible.

Now, for a given word w and the computed vector β one can obtain the value $\hat{P}(w)$ predicted by the regression model as

$$\hat{P}(w) = \beta' f(w).$$

Unfortunately, we cannot guarantee that $P(w)$ is, indeed, a linear function of $f(w)$. If the standard linear regression model fails one can change the representation of the data replacing the feature vector $f(w)$ by its image via a non-linear mapping $\varphi = (\varphi_1, \dots, \varphi_k)$:

$$f(w) \rightarrow \varphi(f(w)) = \langle \varphi_1(f(w)), \dots, \varphi_k(f(w)) \rangle.$$

One of the most common approaches is to use a general quadratic mapping (where the quadratic terms are written in some order)

$$\varphi(\langle c_1, \dots, c_m \rangle) = \langle c_1, \dots, c_m, c_1c_1, c_1c_2, \dots, c_m c_{m-1}, c_m c_m \rangle$$

so

$$f_\varphi(w) = \varphi(f(w))$$

is a vector consisting of components of $f(w)$ and all their pair-wise products written in some order. Now one can apply the standard linear regression model to the vector f_φ :

$$\mathcal{P} = V_\varphi \beta + \epsilon,$$

where $V_\varphi = [f_\varphi(w_1), \dots, f_\varphi(w_n)]$, find the vector of regression coefficients β_φ , and compute the predicted value

$$\hat{P}(w) = \beta_\varphi' f_\varphi(w).$$

We will denote by $f^2(w)$ the quadratic mapping of the feature vector $f(w)$.

One of the possible linear regression classifiers is based on the following decision rule. Given a word $w \in F(X)$ it returns the answer $decide(w)$ according to the following formula:

$$decide(w) = \begin{cases} 1, & \text{if } \hat{P}(w) > \Theta; \\ 0, & \text{otherwise.} \end{cases} \quad (5.2.2)$$

where Θ is a given threshold. However, there is an ambiguity in selection of the parameter Θ in the decision rule (5.2.2).

Here we elected to use the following Bayesian type of the decision rule. Suppose an event $\hat{P}(w) = \alpha$, where $\alpha \in \mathbb{R}$, is observed. We are going to make a prediction on whether $P(w) = 1$ or $P(w) = 0$ based on the estimation of the conditional probabilities

$$Pr(P(w) = 1 \mid \hat{P}(w) = \alpha) \quad \text{and} \quad Pr(P(w) = 0 \mid \hat{P}(w) = \alpha)$$

so, theoretically, the corresponding decision rule is:

$$decide(w) = \begin{cases} 1, & \text{if } Pr(P(w) = 1 \mid \hat{P}(w) = \alpha) > Pr(P(w) = 0 \mid \hat{P}(w) = \alpha); \\ 0, & \text{otherwise.} \end{cases} \quad (5.2.3)$$

Since we cannot compute the conditional probabilities above precisely, we estimate them as follows. We partition the set $\mathbb{R}$ into intervals Δ of equal length and estimate the conditional probabilities:

$$Pr(P(w) = 1 \mid \hat{P}(w) \in \Delta) \quad \text{and} \quad Pr(P(w) = 0 \mid \hat{P}(w) \in \Delta)$$

Using Bayes' formula one can rewrite the probabilities above ($i = 0, 1$):

$$Pr(P(w) = i \mid \hat{P}(w) \in \Delta) = \frac{Pr(\hat{P}(w) \in \Delta \mid P(w) = i)Pr(P(w) = i)}{Pr(\hat{P}(w) \in \Delta)}$$

Therefore

$$Pr(P(w) = 1 \mid \hat{P}(w) \in \Delta) > Pr(P(w) = 0 \mid \hat{P}(w) \in \Delta)$$

if and only if

$$Pr(\hat{P}(w) \in \Delta \mid P(w) = 1)Pr(P(w) = 1) > Pr(\hat{P}(w) \in \Delta \mid P(w) = 0)Pr(P(w) = 0) \quad (5.2.4)$$

The probabilities $Pr(P(w) = 1)$ and $Pr(P(w) = 0)$ are prior probabilities corresponding to the distribution of minimal and non-minimal elements among the inputs given to the classifier.

We have already mentioned that in general situation a randomly chosen element of a free group is Whitehead-minimal with probability 1. This makes the classification task simple if we assume that inputs will be chosen randomly. However, the rate of false positive error (the error of classifying non-minimal element as minimal, see Section 5.2.4) in this case will be very high. The class of non-minimal elements is of the same interest as the class of minimal elements. To avoid bias towards minimal elements, we choose a more conservative approach by choosing equal prior probabilities for both classes.

It is safe to assume that the prior probabilities are equal. Thus the inequality (5.2.4) takes the form

$$Pr(\hat{P}(w) \in \Delta \mid P(w) = 1) > Pr(\hat{P}(w) \in \Delta \mid P(w) = 0)$$

The conditional probabilities above can be estimated from the given training dataset D . For $i = 0, 1$ put

$$d_i(\Delta) = \frac{|\{w \mid \hat{P}(w) \in \Delta, \langle w, i \rangle \in D\}|}{|D|}$$

Then

$$Pr(\hat{P}(w) \in \Delta \mid P(w) = i) \approx d_i(\Delta), \quad i = 0, 1.$$

Finally we can define the following decision rule, which is a variation of the Bayes' decision rule above:

$$decide(w) = \begin{cases} 1, & \text{if } \hat{P}(w) \in \Delta \text{ and } d_1(\Delta) > d_0(\Delta) \text{ for some interval } \Delta; \\ 0, & \text{if } \hat{P}(w) \in \Delta \text{ and } d_0(\Delta) > d_1(\Delta) \text{ for some interval } \Delta. \end{cases} \quad (5.2.5)$$

5.2.4 Evaluation

Test datasets

To test and evaluate the pattern classification system WMIN we generate several test datasets of different type:

S_e : A test set S_e which is generated by the same procedure as for the training set D , but independently of D .

S_R : A test set S_R of (pseudo-) randomly generated elements of $F(X)$. We used the random walk described in Section 3.2.1 to generate S_R .

S_P : A test set S_P of (pseudo-) randomly generated *primitive* elements in $F(X)$. Recall that $w \in F(X)$ is primitive if and only if there exists a sequence of Whitehead automorphisms $t_1 \dots t_l \in \Omega(X)$ such that $xt_1 \dots t_l = w$ for some $x \in X^{\pm 1}$. Elements in S_P are generated by the procedure described in Section 3.2.3, which, roughly speaking, amounts to a random choice of $x \in X^{\pm 1}$ and a random choice of a sequence of automorphisms $t_1 \dots t_l \in \Omega(X)$.

S_{10} : A test set S_{10} which is generated by the procedure used to generate the set S_{NMin} , where the length of the randomly generated automorphism φ_i is chosen uniformly randomly from the set $\{1, \dots, 10\}$, hence the name.

Some comparative characteristics of the generated datasets are given in Table 5.1.

Dataset	size	% min	% non-min	(min,avg,max) word lengths
D	10000	51.9	48.1	(1,541,1202)
S_e	5000	49.5	50.5	(1,542,1200)
S_{10}	5000	48.6	51.4	(1,691,10629)
S_R	5000	98.8	1.2	(1,499,998)
S_P	6000	0	100	(2,30,3443)

Table 5.1: Description of the training and test datasets in F_2 .**Accuracy measure**

Let D_{eval} be a test data set and

$$K = |\{w \mid decide(w) = P(w), w \in D_{eval}\}|$$

be the number of correctly classified elements in D_{eval} . To evaluate the performance of a given pattern classification system we use a simple accuracy measure:

$$A = \frac{K}{|D_{eval}|},$$

which gives the fraction of the correctly classified elements from the test set D_{eval} .

Notice, that the numbers of correctly classified elements follow the Binomial distribution and A can be viewed as an estimate of probability p of a word being classified correctly.

We are interested in constructing a confidence interval for probability p . For binomial variates, exact confidence intervals do not exist in general. One can obtain an approximate $100(1 - \alpha)\%$ confidence interval $[p_S, p_L]$ by solving the following equations for p_S and p_L :

$$\sum_{i=0}^K \binom{|D_{eval}|}{i} p_S^i (1 - p_S)^{|D_{eval}|-i} = \alpha/2,$$

$$\sum_{i=K}^{|D_{eval}|} \binom{|D_{eval}|}{i} p_L^i (1 - p_L)^{|D_{eval}|-i} = \alpha/2$$

for a given α .

As we already mentioned before (see Section 3.3.1) such equations can be solved by re-expressing in terms of the incomplete beta function.

Note, that there are two types of error, called false positive and false negative, that can occur during the classification of minimal elements. A false positive is an error of classifying a non-minimal element as minimal. A false negative error is when a minimal element is classified as a non-minimal element.

In some applications it could be important to maintain one of the errors at the predefined level even if it will cause the second error to increase. In our case we do not give any preference to either of the classes and will expect the rates of the two errors be relatively equal.

5.2.5 Feature selection algorithm

Let $\mathcal{S}$ be a pattern classification system and $\mathbf{P}$ be the corresponding classifier. The performance of the classifier $\mathbf{P}$ often directly depends on the set of features built into $\mathcal{S}$. Sometimes it is possible to reduce the number of features in $\mathcal{S}$ maintaining the same level of classification accuracy of $\mathbf{P}$, and even find more efficient combinations of the given features. The corresponding procedure is called *feature selection*. We give a description of one of possible procedures below.

Let $\mathcal{C}$ be a finite collection of counting functions (see Section 5.2.2). Every sequence $\bar{C} = \langle C_1, \dots, C_l \rangle$ of functions from $\mathcal{C}$ gives rise to the corresponding feature mapping $f_{\bar{C}}$. Denote by $\mathcal{S}_{\bar{C}}$ the pattern classification system obtained from $\mathcal{S}$ by

replacing the feature set in $\mathcal{S}$ by $\bar{C}$. Let $\mathbf{P}_{\bar{C}}$ be the classifier that corresponds to the system $\mathcal{S}_{\bar{C}}$. Every system $\mathcal{S}_{\bar{C}}$ has one and the same test data set D_{eval} and the same accuracy measure A . Denote by $A(f_{\bar{C}}) = A(\bar{C})$ the accuracy of the classifier $\mathbf{P}_{\bar{C}}$ evaluated on the set D_{eval} .

We implement the feature selection algorithm as an iterative greedy procedure. On each iteration i a new feature mapping f_i with the current best evaluation value $A(f_i)$ is selected and added to the set $\mathcal{F}$ of feature mappings constructed before. The procedure stops in at most $|\mathcal{C}|$ iterations. The best overall feature mapping $f^* \in \mathcal{F}$ of minimal length is returned as the output of the procedure. More precisely, the algorithm proceeds as follows:

Iteration 1:

Choose $C_1 \in \mathcal{C}$ such that

$$A(C_1) = \max \{A(C) \mid C \in \mathcal{C}\};$$

Set $f_1 = f_{C_1}$ and $\mathcal{F} = \{f_1\}$

Iteration N :

Suppose feature mappings $f_1, \dots, f_{N-1}$ are constructed and

$$f_{N-1} = f_{\langle C_1, \dots, C_{N-1} \rangle}$$

for some $C_1, \dots, C_{N-1} \in \mathcal{C}$. Choose $C_N \in \mathcal{C} - \{C_1, \dots, C_{N-1}\}$ such that the sequence $\bar{C}_N = \langle C_1, \dots, C_N \rangle$ satisfies the following condition:

$$A(\bar{C}_N) = \max \{A(\bar{C}) \mid \bar{C} = \langle C_1, \dots, C_{N-1}, C \rangle, C \in \mathcal{C} - \{C_1, \dots, C_{N-1}\}\}.$$

Put $f_N = f_{\tilde{C}_N}$ and $\mathcal{F} = \mathcal{F} \cup \{f_N\}$.

If $N = |\mathcal{C}|$ then STOP.

Put

$$A_{max} = \max \{A(f) \mid f \in \mathcal{F}\}$$

Select the mapping $f^* \in \mathcal{F}$ such that

$$A(f^*) \in I(A_{max}),$$

where $I(A_{max})$ is the 95 percent confidence interval described in Section 5.2.4, and f^* has the smallest possible length among all such feature mappings.

Observe, that the feature selection procedure described above does not check all possible feature mappings that can be built from the counting functions from $\mathcal{C}$. There would be too many of them for even reasonably small sets $\mathcal{C}$. Instead, it makes at most $|\mathcal{C}|$ iterations, though each iteration could be time consuming since it requires evaluation of the current classifier $\mathbf{P}_{\tilde{C}}$.

5.3 Experiments in F_2

5.3.1 Evaluating classifiers

In this section we present results of evaluation of classifiers $\mathbf{P}_f$ on the test dataset S_e when f runs over a particular set of feature mappings. By $A(f)$ we denote the accuracy of the classifier $\mathbf{P}_f$.

Let

$$f_1(w) = \frac{1}{|w|} < C(w, v) \mid v \in V_2 >$$

be the feature mapping discussed in Section 5.2.2 and $f_{WG}(w)$ be the feature mapping based on the graph $WG(w)$. Recall that in view of the characterization of feature mappings as corresponding to the subgraphs of the graph $\Gamma(w)$ (see the end of Section 5.2.2) the mapping f_1 corresponds to the subgraph $\Gamma_1(w)$ which is a directed analog of the Whitehead's graph of w . The accuracy of the classifiers $\mathbf{P}_1 = \mathbf{P}_{f_1}$ and $\mathbf{P}_{WG} = \mathbf{P}_{f_{WG}}$ is over 95% (Table 5.2), which is quite good, but leaves some room for improvements. Notice that $\mathbf{P}_1$ and $\mathbf{P}_{WG}$ give almost identical results. However, since $|f_{WG}| < |f_1|$ classifier $\mathbf{P}_{WG}$ is more preferable. Consider the following feature mappings which correspond to various subgraphs of the graph $\Gamma(w)$:

$$\begin{aligned} f_2(w) &= \frac{1}{|w|} \langle C(w, x_1 v x_2) \mid x_1, x_2 \in X^{\pm 1}, v \in V_1 \rangle; \\ f_3(w) &= \frac{1}{|w|} \langle C(w, x_1 v x_2) \mid x_1, x_2 \in X^{\pm 1}, v \in V_2 \rangle; \\ f_4(w) &= \frac{1}{|w|} \langle C(w, x_1 v x_2) \mid x_1, x_2 \in X^{\pm 1}, v \in V_3 \rangle; \\ f_5(w) &= \frac{1}{|w|} \langle C(w, x_1 v x_2) \mid x_1, x_2 \in X^{\pm 1}, v \in W_1 \rangle; \\ f_6(w) &= \frac{1}{|w|} \langle C(w, x_1 v x_2) \mid x_1, x_2 \in X^{\pm 1}, v \in W_3 \rangle. \end{aligned}$$

The results of evaluation of the classifiers $\mathbf{P}_i = \mathbf{P}_{f_i}, i = 1, \dots, 6$ on S_e are given in Table 5.2a. We also test classifiers $\mathbf{P}_6$, and $\mathbf{P}_{WG}$ on the datasets S_R, S_{10} , and S_P . The results of these tests are given in Table 5.2b.

One can see that the classifier $\mathbf{P}_6$ is robust and performs well even on datasets which are essentially different from the training dataset D . The classifier $\mathbf{P}_{WG}$ has difficulties with primitive elements.

Next, we repeat the set of experiments with the classifiers $\mathbf{P}_{WG}^2, \mathbf{P}_1^2, \dots, \mathbf{P}_6^2$, where $\mathbf{P}_a^2$ is a classifier based on the quadratic feature mapping f_a^2 . The results are presented

	$A(f_1)$	$A(f_2)$	$A(f_3)$	$A(f_4)$	$A(f_5)$	$A(f_6)$	$A(f_{WG})$
$ w > 0$	.954	.968	.926	.869	.977	.980	.951
$ w > 4$	.957	.969	.927	.870	.977	.981	.953
$ w > 100$	.975	.984	.947	.893	.992	.994	.975

a) Performance of the classifiers $\mathbf{P}_1, \dots, \mathbf{P}_6, \mathbf{P}_{WG}$ on the set S_e .

	S_{10}		S_R		S_P	
	$A(f_6)$	$A(f_{WG})$	$A(f_6)$	$A(f_{WG})$	$A(f_6)$	$A(f_{WG})$
$ w > 0$	.981	.829	.978	.954	.879	.583
$ w > 4$	.982	.830	.979	.956	.922	.580
$ w > 100$	.994	.844	.993	.983	1.00	.550

b) Performance of the classifiers $\mathbf{P}_6, \mathbf{P}_{WG}$ on the test datasets S_R, S_{10}, S_P .

Table 5.2: Performance of the classifiers based on linear mappings in F_2 .

in Table 5.3. We can see that the addition of quadratic and interaction terms to the features improves the performance of all classifiers. In particular classifier $\mathbf{P}_{WG}^2$ based on the shortest feature mapping performs as well as classifier $\mathbf{P}_6^2$. Notice perfect accuracy of $\mathbf{P}_{WG}^2$ on large words on all test datasets.

The estimated conditional probabilities for $\mathbf{P}_6$ and $\mathbf{P}_{WG}^2$ (which come from the Bayes' decision rule, see Section 5.2.3) are presented in Figure 5.1. Clearly, the classes of minimal and non-minimal elements are separated around 0.5 with a small overlap. So the regression works perfectly with the threshold $\Theta = 0.5$. The area of the overlap corresponds to the classification error. The larger the overlap the more probable that a classifier will make a wrong decision. We can see that the classification error of $\mathbf{P}_{WG}^2$ is much smaller which explains its by far better performance.

Below we summarize the results of our experiments in the following conclusions:

Conclusion 5.3.1. *The accuracy of the classifiers dramatically increases when one adds quadratic and interaction terms to the feature mappings;*

	$A(f_1^2)$	$A(f_2^2)$	$A(f_3^2)$	$A(f_4^2)$	$A(f_5^2)$	$A(f_6^2)$	$A(f_{WG}^2)$
$ w > 0$	.995	.984	.964	.912	.995	.995	.997
$ w > 4$	.996	.985	.965	.912	.995	.995	.998
$ w > 100$	1.00	.997	.989	.945	1.00	1.00	1.00

a) Performance of the classifiers $\mathbf{P}_1^2, \dots, \mathbf{P}_6^2, \mathbf{P}_{WG}^2$ on the set S_e .

	S_{10}		S_R		S_P	
	$A(f_6^2)$	$A(f_{WG}^2)$	$A(f_6^2)$	$A(f_{WG}^2)$	$A(f_6^2)$	$A(f_{WG}^2)$
$ w > 0$	.993	.997	.990	.994	.919	1.00
$ w > 4$	.994	.998	.992	.995	.924	1.00
$ w > 100$	1.00	1.00	1.00	1.00	.970	1.00

b) Performance of the classifiers $\mathbf{P}_6^2, \mathbf{P}_{WG}^2$ on the test datasets S_R, S_{10}, S_P .

Table 5.3: Performance of the classifiers based on quadratic mapping in F_2 .

Conclusion 5.3.2. *The classifier $\mathbf{P}_{WG}^2$ is the best so far, it is remarkably reliable;*

Conclusion 5.3.3. *Very short words are difficult to classify (perhaps, because they do not provide sufficient information for the classifiers);*

5.3.2 Feature selection and analysis of the pattern recognition systems

In this section we are looking for a feature mapping which is at least as effective as f_{WG}^2 , but contains considerably less features. Observe, that f_6 , as a vector, consists of 60 components (features) and f_{WG}^2 contains $12^2 = 144$. In the search for the most effective feature mapping we apply the Feature Selection Algorithm from Section 5.2.5 to the set of all counting functions involved in f_6 . Put

$$\mathcal{C} = \{C(w, xvy) \mid x, y \in X^{\pm 1}, v \in F(X), W_3\},$$

so counting functions from $\mathcal{C}$ correspond to edges of the graph $\Gamma_6(w)$.

We were very surprised when the Feature Selection Algorithm, applied to the set $\mathcal{C}$, found a feature mapping based on only two counting functions:

$$f^*(w) = \frac{1}{|w|} \langle C(w, a^{-1}b), C(w, b^{-1}a) \rangle$$

where $X = \{a, b\}$.

The classifier $\mathbf{P}_* = \mathbf{P}_{f^*}$ based on the feature mapping f^* showed a competitive performance when tested on all datasets. The results of comparison of $\mathbf{P}_*$ with $\mathbf{P}_{WG}^2$ and $\mathbf{P}_6$ are presented in Table 5.4. The estimated conditional probabilities for $\mathbf{P}_*$ are given on Figure 5.2.

One can see that non-minimal elements in S_e are divided into two clusters M_1 and M_2 such that the regression values for the class of the minimal elements lay in between the regression values for elements in M_1 and M_2 . This shows that the linear regression cannot predict correctly values $P(w)$ of the predicate P for $w \in S_e$. Indeed, the standard threshold-based decision rule (5.2.2) will always give an error, at least, in 25% of trials, no matter what threshold value is chosen.

However, there is an obvious separation between minimal and non-minimal elements on Figure 5.2 and the Bayesian decision rule (5.2.5) was able to catch it. Since $f^*(w)$ is a two-dimensional vector, one can plot points $f^*(w), w \in S_e$, on a Euclidean plane. On Figure 5.3 we show scatter plot for f^* . Again, one can clearly see three groups of points. The one in the middle corresponds to the class of minimal elements, two others formed by non-minimal elements.

What is left is to give an explanation of the unexpected partition

$$NM(S_e) = M_1 \cup M_2$$

of the class $NM(S_e)$ of non-minimal elements from S_e into the clusters M_1 and M_2

	$A(f_{WG}^2)$	$A(f_6)$	$A(f^*)$
$ w > 0$	.997	.980	.987
$ w > 4$	.998	.981	.989
$ w > 100$	1.00	.994	.993

a) S_e

	$A(f_{WG}^2)$	$A(f_6)$	$A(f^*)$
$ w > 0$	.997	.981	.981
$ w > 4$	.998	.982	.983
$ w > 100$	1.00	.994	.993

b) S_{10}

	$A(f_{WG}^2)$	$A(f_6)$	$A(f^*)$
$ w > 0$	.994	.978	.967
$ w > 4$	.995	.979	.975
$ w > 100$	1.00	.993	.992

a) S_R

	$A(f_{WG}^2)$	$A(f_6)$	$A(f^*)$
$ w > 0$	1.00	.879	.945
$ w > 4$	1.00	.922	.922
$ w > 100$	1.00	1.00	.979

a) S_P .Table 5.4: Comparative results for $\mathbf{P}_*$.

reflected on Figure 5.2 for the conditional probabilities for f^* .

Direct inspection of the two clusters M_1 and M_2 shows that the clustering was based on the type of elementary Whitehead automorphisms that reduce the length of a given element from $NM(S_e)$. More precisely, Tables 5.5 show that the set of elementary Nielsen automorphisms T can be partitioned into two subsets $T = T_1 \cup T_2$,

where

$$T_1 = \left\{ \begin{pmatrix} a \rightarrow ba \\ b \rightarrow b \end{pmatrix}, \begin{pmatrix} a \rightarrow ab \\ b \rightarrow b \end{pmatrix}, \begin{pmatrix} a \rightarrow a \\ b \rightarrow ab \end{pmatrix}, \begin{pmatrix} a \rightarrow a \\ b \rightarrow ba \end{pmatrix} \right\}$$

$$T_2 = \left\{ \begin{pmatrix} a \rightarrow b^{-1}a \\ b \rightarrow b \end{pmatrix}, \begin{pmatrix} a \rightarrow ab^{-1} \\ b \rightarrow b \end{pmatrix}, \begin{pmatrix} a \rightarrow a \\ b \rightarrow a^{-1}b \end{pmatrix}, \begin{pmatrix} a \rightarrow a \\ b \rightarrow ba^{-1} \end{pmatrix} \right\}$$

such that automorphisms from T_i are most likely to reduce the length of elements from the cluster M_i , and very rarely reduce the length of elements from the other cluster. Therefore, the classifier $\mathbf{P}_*$ not only solves the minimality classification problem for elements $w \in F(X)$, but it predicts length reducing automorphisms for a given $w \in F(X)$! If true this would reduce by half the search of a minimal element w_{min} for a given w . To verify this phenomena we look at the distributions of the conditional probabilities for f^* on the test datasets S_{10} and S_P . Figure 5.4 presents the conditional probabilities for f^* on the test datasets S_{10} .

One can see a similar decomposition of the sets $NM(S_{10})$ of non-minimal elements in S_{10} into two clusters M_1 and M_2 . A similar clustering takes place also for the set S_P (even though for S_P the clustering structure is more complicated and we omit the picture).

Table 5.5 shows that the sets of Nielsen automorphisms T_1 and T_2 play similar role in clustering of $NM(S_{10})$ and $NM(S_P)$ as in $NM(S_e)$, thus experimentally confirming the phenomena we observed for $NM(S_e)$.

Conclusion 5.3.4. *The Feature Selection Algorithm is useful, it found by far the most economic and effective feature mapping f^* ;*

Conclusion 5.3.5. *The classifier $\mathbf{P}_*$ not only solves the minimality classification problem, as a bonus it also predicts what are the most likely automorphisms which reduce the length of a given non-minimal element $w \in F(X)$.*

Results of Conclusion 5.3.5 suggest that a more general cluster analysis of Whitehead non-minimal words might be useful. In Chapter 6 we describe the results of the analysis and its applications in developing heuristics for the search procedure which solves the Length Reduction Problem.

5.4 Experiments with Regression in F_3 , F_4 and F_5

The free group on two generators is the first "non-trivial" free group that we explore. In Lemma 2.4.2 we showed that the set of Nielsen transformations N_2 is sufficient to solve Whitehead minimization problem in F_2 . This makes the free group F_2 a very special instance, and in case of the Whitehead's problem the easiest one. It is important to show that WMIN performs well in the free groups with ranks larger than 2.

Definition 5.4.1. *The generating set $X(w)$ of an element $w \in F(X)$ is the set of all generators from X that actually occur in w :*

$$X(w) = \{x \mid x^{\pm 1} \in w, x \in X\}.$$

The size of the set $X(w)$ is called the rank of the word w and is denoted by $\rho(w)$.

Before we describe the experiments, we would like to make the following observation.

Lemma 5.4.1. *Let w be a minimal word in free group $F(X)$, then it is also minimal in the free group $F(X(w))$*

Proof. Consider a subgroup $S(X(w))$ of a free group $F(X)$ with the generating set $X(w)$. It is well known that S forms a free group $F(X(w))$ with a bases $X(w)$.

Obviously w is an element of S and, therefore, $w \in F(X(w))$. Now, suppose that w is not minimal in $F(X(w))$, then by Theorem 2.1.3, there exists a Whitehead automorphism $\varphi \in \text{Aut}(F(X(w)))$ such that $|\varphi(w)| < |w|$. Let φ be in the form

$$x \rightarrow v_x, \quad x \in F(X), \quad v_x \in F(X(w)).$$

Now consider an extended map

$$\varphi' : F(X) \rightarrow F(X)$$

defined by mapping every $x \in X$ such that

$$x \rightarrow \begin{cases} v_x, & x \in X(w), \\ x, & \text{otherwise.} \end{cases}$$

It is easy to see from the definition of Whitehead automorphisms that map φ' is a Whitehead automorphism from $\text{Aut}(F(X))$. It is easy to see that

$$\varphi(w) = \varphi'(w).$$

Therefore we have $\varphi'(w) \in \text{Orb}(w) \subset F(X)$ and $|\varphi'(w)| < |w|$ which contradicts the lemma's condition that w is minimal in $F(X)$. ■

Lemma 5.4.1 allows us to transform the problem of deciding whether a word w is minimal into the free group $F(X(w))$. Obviously, in case of $X(w) \subset X$ solving in $F(X(w))$ is preferable because it requires less computational and memory resources. Throughout the rest of the report, we will consider a problem of classifying a particular word w in the corresponding free group $F(X(w))$.

We repeat experiments described in the previous section with datasets of elements from free groups F_3 , F_4 and F_5 . The classifiers were trained on the training sets

$$\bar{D}_n = D_n - \{w \mid \rho(w) < n\},$$

where $D_n \subset F_n$ is the training set generated using procedure described in Section 5.2.1 and $n = 3, 4, 5$.

The evaluation was performed using test sets $\bar{S}_e, \bar{S}_{10}, \bar{S}_R, \bar{S}_P$, where for $S_A \in F_n$, $A = e, 10, R, P$

$$\bar{S}_A = S_A - \{w \mid \rho(w) < n\}$$

and S_A is one of the test sets described in Section 5.2.4.

For the convenience we will further omit “ $\bar{\cdot}$ ” from the notation of the datasets.

The datasets were generated for each of the groups F_3, F_4 and F_5 . Table 5.6 shows some of the characteristics of the sets used in the experiments.

Experiments with the free group F_2 showed that the classifier $\mathbf{P}_{WG}^2$ based on the quadratic mapping of the feature mapping f_{WG} produces the best over all accuracy. The same happened to be true for the free groups F_3, F_4, F_5 . The best results were achieved when we used the quadratic regression on the features constructed from the weights of the Whitehead graph. $\mathbf{P}_{WG}^2$ produced results with the accuracy shown in Table 5.7.

Conclusion 5.4.2. *The experiments show that it is possible to build a system for classifying the Whitehead minimal elements, which gives reliable results with high accuracy for the groups with ranks greater than two.*

For a given word $w \in F_n$ there are total of $2n(2n - 1)^1$ edge weights in the graph $WG(w)$. Therefore, the classifier $\mathbf{P}_{WG}^2$ is required to compute $(2n(2n - 1))^2$ regression coefficients. This constitutes two main problems. First, in order to compute the coefficients we have to multiply and decompose matrices of size equal to the number of the coefficients itself. For large n , the straightforward computation of such matrices

¹There are $2n$ nodes and $2n(2n - 1)$ edges, since graph $WG(w)$ does not contain self-loops.

might be impossible due to the memory size restrictions. Another problem is due to the fact that the number of observations in the training set needs to be about 100 times more than the number of the coefficients to be estimated. When n is large (for $n = 10$ the required number of observations is about 14,440,000) it is a significant practical limitation, especially when the data generation is time consuming.

In the next section we apply the Support Vector Machines to recognize minimal elements in free groups of large ranks.

5.5 Support Vector Machine Classifier

In Section 1.3.4 we have stressed that one of the main attractive features of the Support Vector Machines is their ability to employ non-linear mapping without essential increase in the number of parameters to be estimated and, therefore, in computation time. In this section we apply SVMs to the task of recognizing the Whitehead-minimal elements in free groups of large ranks.

To show that the performance of Support Vector Machines is acceptable for free groups with large ranks we run experiments with groups of ranks 3,5,10,15,20. For each group we construct the training set D and test sets S_e , S_{10} , S_R , S_P using procedures described previously. Some statistics of the datasets are given in Table 5.8.

Experiments were repeated with the following types of kernel functions:

K^1 : linear;

K^2 : quadratic $(b\mathbf{x}_i' \cdot \mathbf{x}_j + c)^2$;

K^3 : polynomial of degree 3 $(b\mathbf{x}_i' \cdot \mathbf{x}_j + c)^3$;

K^4 : polynomial of degree 4 $(b\mathbf{x}_i' \cdot \mathbf{x}_j + c)^4$;

K^e : Gaussian $e^{-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2}$,

where $\mathbf{x}_i, \mathbf{x}_j$ are the feature vectors obtained with mapping f_{WG} and $\mathbf{x}_i \cdot \mathbf{x}_j$ is the inner product of $\mathbf{x}_i$ and $\mathbf{x}_j$. See section 1.3.4 for more detailed discussion on kernel properties and their correspondence to Support Vector Machines.

The results of the experiments are presented in Table 5.9. It shows that SVM with an appropriate kernel performs well not only on free groups of small ranks but on groups of large ranks as well. The experiments confirmed observations, made previously, that classes of minimal and non-minimal words are not linearly separable. Moreover, once the rank and, therefore dimensionality of the feature space grows, quadratic mapping, successfully used to recognize minimal elements in free groups F_r , $r = 2, 3, 4, 5$ does not guarantee the high classification accuracy. As one might expect, the accuracy increases when the degree of the polynomial mapping increases. Nevertheless, even with the polynomial kernel K^4 of the degree $d = 4$, the Support Vector Machine is not able to perform accurate classification in groups F_{15} and F_{20} . However, the Gaussian kernel produces stable and accurate results for all test datasets, including sets of elements in free groups of large ranks. This indicates that points in one of the classes (minimal or non-minimal) are compactly distributed in the feature space and can be accurately described as a Gaussian. We also can observe that the Gaussian representation can be applied to only one of the classes. If the opposite was true, then the problem of separating the two classes would be much simpler and at least the quadratic mapping should have been as accurate as K^e .

We conclude this section with the following

Conclusion 5.5.1.

1. *With an appropriate kernel function Support Vector Machines approach performs very well in the task of classification of Whitehead-minimal words in free groups of various ranks, including groups of large ranks.*
2. *The best over all results are obtained with the Gaussian kernel K^e . This indicates that one of the classes is compact and can be bounded by a Gaussian function.*
3. *The regression approach is still would be preferable for groups of small ranks due to its simplicity and smaller resource requirements. However, the SMVs should be used for groups of larger ranks where the size of the training sets required to perform regression with non-linear preprocessing mapping becomes practically intractable.*

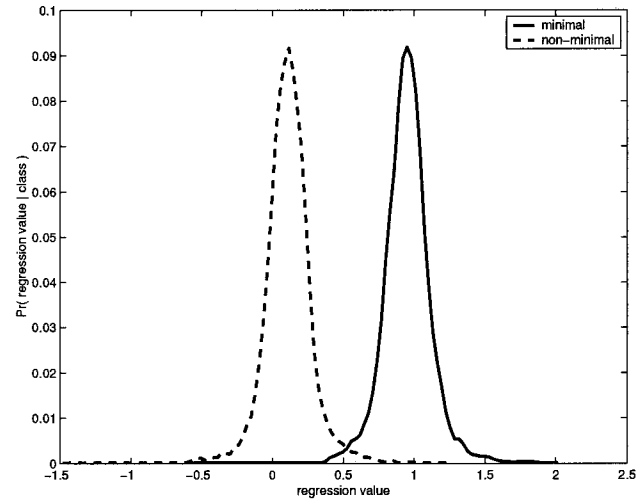
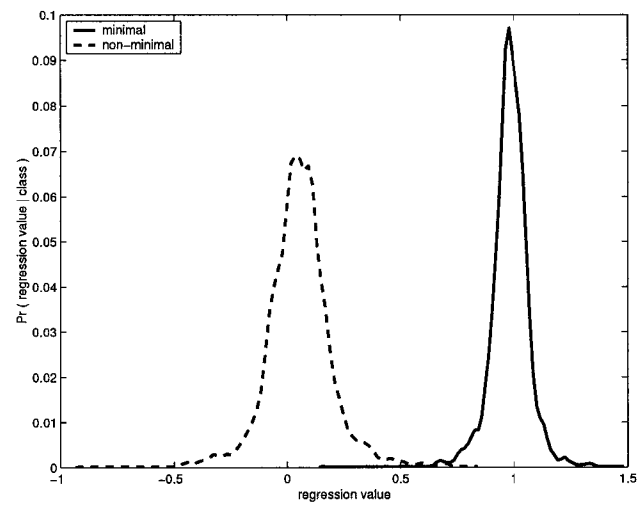
a) $P_6;$ b) P_{WG}^2

Figure 5.1: Conditional probabilities.

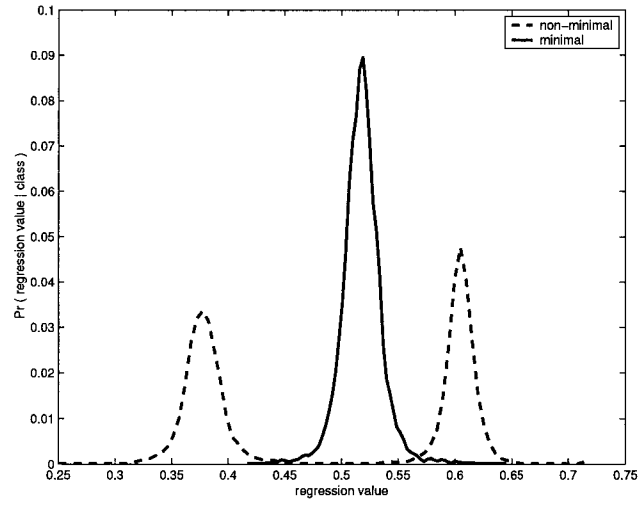


Figure 5.2: Conditional probabilities for $\mathbf{P}_*$.

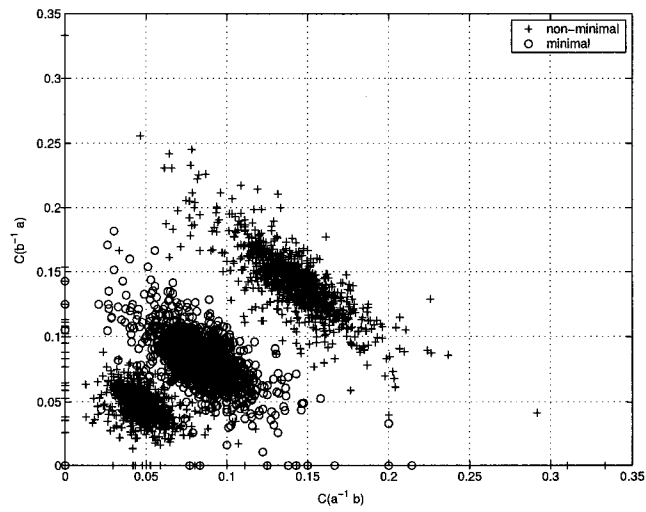


Figure 5.3: Scatter plot of points $f^*(w)$, $w \in S_e$.

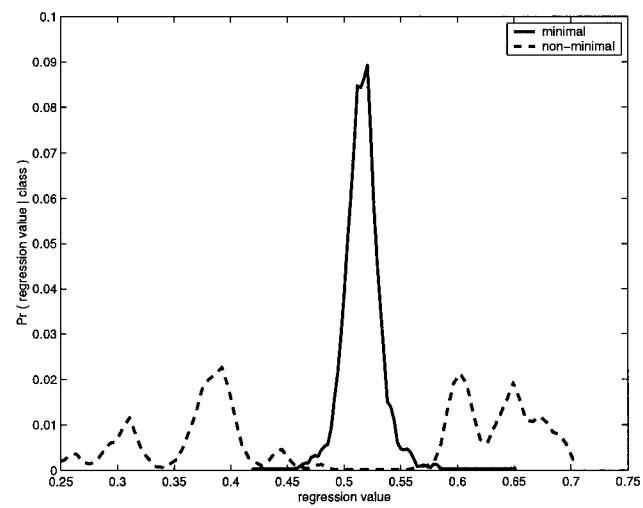


Figure 5.4: Conditional probabilities for $\mathbf{P}_*$ on S_{10} .

	$NM(S_e)$		$NM(S_{10})$		$NM(S_P)$	
	M_1	M_2	M_1	M_2	M_1	M_2
$a \rightarrow ba, b \rightarrow b$	.7152	.0008	.7480	.0008	.7671	.0406
$a \rightarrow ab, b \rightarrow b$	.7136	.0023	.7488	.0023	.7671	.0406
$a \rightarrow a, b \rightarrow ab$	.7522	.0000	.7457	.0023	.7663	.0543
$a \rightarrow a, b \rightarrow ba$	.7458	.0038	.7417	.0031	.7663	.0543
$a \rightarrow b^{-1}a, b \rightarrow b$	.0016	.7320	.0000	.7567	.0000	.6996
$a \rightarrow ab^{-1}, b \rightarrow b$	.0016	.7328	.0008	.7559	.0000	.6996
$a \rightarrow a, b \rightarrow a^{-1}b$	.0000	.7199	.0008	.7291	.0000	.6924
$a \rightarrow a, b \rightarrow ba^{-1}$	.0008	.7184	.0000	.7322	.0000	.6924

a) all elements;

	$NM(S_e)$		$NM(S_{10})$		$NM(S_P)$	
	M_1	M_2	M_1	M_2	M_1	M_2
$a \rightarrow ba, b \rightarrow b$	.7264	.0000	.7588	.0000	.7614	.0000
$a \rightarrow ab, b \rightarrow b$	.7264	.0000	.7588	.0000	.7614	.0000
$a \rightarrow a, b \rightarrow ab$	.7582	.0000	.7502	.0000	.7386	.0000
$a \rightarrow a, b \rightarrow ba$	.7582	.0000	.7502	.0000	.7386	.0000
$a \rightarrow b^{-1}a, b \rightarrow b$	.0000	.7487	.0000	.7683	.0000	.7368
$a \rightarrow ab^{-1}, b \rightarrow b$	.0000	.7487	.0000	.7683	.0000	.7368
$a \rightarrow a, b \rightarrow a^{-1}b$	.0000	.7334	.0000	.7479	.0000	.8092
$a \rightarrow a, b \rightarrow ba^{-1}$	.0000	.7334	.0000	.7479	.0000	.8092

b) $|w| > 100$.

Table 5.5: Fraction of elements in $NM(S_e)$, $NM(S_{10})$ and $NM(S_P)$ reduced by automorphisms from T_1 and T_2 .

Dataset	size	% min	% non-min	(min,avg,max) word lengths
D	20000	49.1	50.9	(3,558.2,1306)
S_e	5000	48.9	51.1	(3,559,1292)
S_{10}	5000	49.1	50.9	(3,1016.5,13381)
S_R	5000	98.3	1.7	(3,501.2,999)
S_P	3850	0.0	100.0	(3,194.7,8719)

a) F_3 ;

Dataset	size	% min	% non-min	(min,avg,max) word lengths
D	20000	48.7	51.3	(4,570.9,1366)
S_e	5000	49.6	50.4	(4,570,1344)
S_{10}	5000	49.5	50.5	(5,1398.7,34609)
S_R	5000	97.7	2.3	(5,503.1,999)
S_P	3250	0.0	100.0	(4,262.0,13071)

b) F_4 ;

Dataset	size	% min	% non-min	(min,avg,max) word lengths
D	20000	48.5	51.5	(5,581.3,1388)
S_e	5000	49.2	50.8	(8,583.7,1382)
S_{10}	5000	48.0	52.0	(7,1693.22,28278)
S_R	5000	97.2	2.8	(6,504.2,999)
S_P	2900	0.0	100.0	(5,656.9,22430)

c) F_5 ;Table 5.6: Description of the training and test datasets in free groups F_3 , F_4 and F_5 .

	F_3	F_4	F_5
$ w > 1$	.995	.995	.989
$ w > 100$	1.00	1.00	1.00

a) S_e ;

	F_3	F_4	F_5
$ w > 1$	.994	.990	.987
$ w > 100$	1.00	1.00	1.00

b) S_{10} ;

	F_3	F_4	F_5
$ w > 1$	.988	.985	.987
$ w > 100$	1.00	1.00	1.00

c) S_R ;

	F_3	F_4	F_5
$ w > 1$	.994	.991	.995
$ w > 100$	1.00	1.00	1.00

d) S_P .

Table 5.7: Performance of the classifier $\mathbf{P}_{WG}^2$ in free groups F_3 , F_4 and F_5 .

Dataset	size	% min	% non-min	(min,avg,max) word lengths
D	9660	48.9	51.1	(26,617.4,1461)
S_e	4811	49.2	50.8	(26,619.7,1443)
S_{10}	4837	49.5	50.5	(29,2589.8,65274)
S_R	4867	96.5	3.5	(18,512.7,999)
S_P	165	0.0	100.0	(12,150.8,1459)

a) F_{10} ;

Dataset	size	% min	% non-min	(min,avg,max) word lengths
D	9357	49.5	50.5	(41,635.3,1472)
S_e	4685	49.2	50.8	(40,642.5,1462)
S_{10}	4722	49.7	50.3	(46,3056.6,53422)
S_R	4755	95.3	4.7	(26,523.8,999)
S_P	870	0.0	100.0	(28,1109.3,4981)

b) F_{15} ;

Dataset	size	% min	% non-min	(min,avg,max) word lengths
D	9144	49.6	50.4	(47,658.3,1488)
S_e	4576	49.3	50.7	(48,659.8,1484)
S_{10}	4597	49.1	50.9	(64,3351.4,68316)
S_R	4643	94.0	6.0	(48,534.9,999)
S_P	182	0.0	100.0	(66,945.1,4762)

c) F_{20} ;

Table 5.8: Description of the training and test datasets in free groups F_{10} , F_{15} and F_{20} .

	All elements					Elements with $ w > 100$				
Kernel	F_3	F_5	F_{10}	F_{15}	F_{20}	F_3	F_5	F_{10}	F_{15}	F_{20}
K^1	.844	.805	.729	.676	.644	.859	.810	.738	.680	.648
K^2	.995	.978	.881	.782	.710	.999	.977	.880	.792	.711
K^3	.996	.988	.962	.888	.772	1.00	.996	.968	.894	.773
K^4	.996	.989	.984	.951	.832	1.00	.997	.991	.956	.834
K^e	.995	.986	.982	.988	.990	1.00	.999	.999	.998	.995

a) accuracy evaluated on the set S_e ;

	All elements					Elements with $ w > 100$				
Kernel	F_3	F_5	F_{10}	F_{15}	F_{20}	F_3	F_5	F_{10}	F_{15}	F_{20}
K^1	.806	.783	.760	.670	.676	.818	.798	.768	.717	.687
K^2	.993	.988	.885	.811	.750	.997	.971	.893	.814	.751
K^3	.994	.993	.969	.893	.809	1.00	.998	.976	.897	.810
K^4	.995	.993	.993	.954	.866	1.00	.998	.997	.957	.870
K^e	.995	.993	.985	.986	.989	1.00	1.00	.999	.999	.994

b) accuracy evaluated on the set S_{10} ;

	All elements					Elements with $ w > 100$				
Kernel	F_3	F_5	F_{10}	F_{15}	F_{20}	F_3	F_5	F_{10}	F_{15}	F_{20}
K^1	.880	.833	.743	.710	.691	.915	.887	.768	.729	.727
K^2	.990	.986	.890	.812	.754	.999	.969	.903	.830	.790
K^3	.991	.991	.961	.893	.824	1.00	.996	.985	.911	.842
K^4	.992	.991	.973	.940	.867	1.00	.997	.999	.970	.883
K^e	.990	.986	.980	.973	.970	1.00	1.00	.999	.989	.973

c) accuracy evaluated on the set S_R ;

	All elements					Elements with $ w > 100$				
Kernel	F_3	F_5	F_{10}	F_{15}	F_{20}	F_3	F_5	F_{10}	F_{15}	F_{20}
K^1	.732	.798	.770	.694	.610	.674	.682	.769	.690	.612
K^2	.999	.997	.824	.722	.610	.993	.993	.785	.723	.626
K^3	1.00	1.00	.915	.777	.632	1.00	1.00	.877	.756	.648
K^4	1.00	1.00	.982	.821	.659	1.00	1.00	.985	.816	.665
K^e	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

d) accuracy evaluated on the set S_P .

Ranks 10-20 have to be reevaluated

Table 5.9: Performance of the Support Vector Machine classifier in free groups F_3 , F_5 , F_{10} , F_{15} and F_{20} .

Chapter 6

Search Heuristics for the Whitehead Minimization Problem

6.1 Introduction

Recall that the worst case complexity of the algorithm DWR on an input word $w \in F_n$ is bounded from above by $cA_n|u|^2$, where A is the number of Whitehead automorphisms in Ω_n , which is exponential in the rank n . Therefore, in the worst case scenario, the algorithm seems to be impractical for free groups with large ranks. One can try to improve on the average number of steps which takes to find a length-reducing automorphism for a given non-minimal element from F_n .

In this chapter we give some empirical evidence that using smart strategies in selecting Whitehead automorphisms in Procedure 2.2.1 one can dramatically improve (in the orders of magnitude) the average complexity of the Length-Reduction Problem in terms of the rank of a free group. We define several heuristics to speed up the procedure on most inputs by imposing particular orderings of automorphisms in ELR procedure.

In the first sections of the chapter we give a description of the search heuristics

and try to estimate their expected performance on randomly generated sets of non-minimal elements. In Section 6.6 we give experimental results evaluating efficiency of different heuristic strategies on sets of primitive elements.

6.2 Nielsen first

The first heuristic comes from a very naive approach: replace the set of Whitehead automorphisms Ω_n in the length reduction routine ELR by the set of Nielsen automorphisms N_n and denote the resulting routine by ELR_N . Since the size of N_n is quadratic and the size of Ω_n is exponential in the rank of F_n , the algorithm ELR_N may give a real speedup in computations. However, we already have mentioned (see Section 2.4) that Whitehead's theorem does not hold after replacement of Ω_n by N_n . Therefore, the algorithm ELR_N will not give the correct answer at least on some inputs. But this is not the end of the story, now the question is *how often the length reduction routine ELR_N gives the correct answer?*

To get some insights, we perform a simple experiment. We use the following test sets of randomly generated cyclically reduced non-minimal elements from free groups F_3 , F_4 and F_5 :

S_1 : set of cyclically reduced non-minimal elements of Whitehead complexity 1;

S_{10} : set of cyclically reduced non-minimal elements of Whitehead complexity at most 10;

S_P : set of cyclically reduced primitive elements.

Some characteristics of the sets are given in Table 6.1.

Dataset	Size	Min. length	Avg. length	Max. length	Std. deviation
S_1	10143	3	605.8	1306	359.3
S_{10}	2535	3	1507.9	13381	1527.9
S_P	5645	3	1422.1	143020	5379.0

a) F_3 ;

Dataset	Size	Min. length	Avg. length	Max. length	Std. deviation
S_1	10176	4	629.3	1366	374.9
S_{10}	2498	5	2273.7	34609	2679.1
S_P	5741	4	4785.3	763650	19266.4

b) F_4 ;

Dataset	Size	Min. length	Avg. length	Max. length	Std. deviation
S_1	10165	5	650.6	1388	385.4
S_{10}	2566	7	2791.1	28278	3234.9
S_P	3821	5	2430.5	160794	6491.0

c) F_5 ;

Table 6.1: Description of the test sets of non-minimal elements in free groups F_3 , F_4 and F_5 .

For each test set S we count the fraction of elements from S which have a length-reducing Nielsen automorphism. The results of the computations together with the corresponding 95% confidence intervals are given in Table 6.2. We can see that most of the words can be reduced by Nielsen automorphisms.

Based on these experiments one can speculate that with very high probability Nielsen automorphisms can reduce the length of a given non-minimal element in F . More precisely, we state the following

Conjecture 6.2.1. *Let S_{NMin}^l be the set of all non-minimal elements in F of length l and $NS_{NMin}^l \subset S_{NMin}^l$ the subset of elements which have Nielsen length-reducing*

	F_3	F_4	F_5
S_1	0.998 [0.9969,0.9988]	0.997 [0.9957,0.9980]	0.998 [0.9969,0.9988]
S_{10}	0.999 [0.9968,0.9998]	0.999 [0.9968,0.9998]	0.998 [0.9954,0.9993]
S_P	0.999 [0.9978,0.9996]	1.000 [0.9993,1.0000]	1.000 [0.9990,1.0000]

Table 6.2: Fraction of elements in the test sets S_1 , S_{10} and S_P with length-reducing Nielsen automorphisms.

automorphisms. Then

$$\lim_{l \rightarrow \infty} \frac{|NS_{NMin}^l|}{|S_{NMin}^l|} = 1.$$

Our first heuristic is based on this conjecture and simply suggests to try Nielsen automorphisms first in the routine ELR, i.e., in this case we assume that in the fixed listing of automorphisms of Ω_n the automorphisms from N_n come first. We refer to this heuristic as *Nielsen First* and denote the corresponding Length Reduction Routine and the Whitehead Reduction algorithm (with respect to this ordering of Ω_n) by ELR_{NF} and DWR_{NF} .

Given that Conjecture 6.2.1 is true, we expect DWR_{NF} to perform much better than DWR on average.

6.3 Cluster analysis

According to the heuristic NF one has to apply Nielsen automorphisms to a given input w in some fixed order, which is independent of the word w . Intuitively, we expect some automorphisms to be more likely to reduce the length of a given word than the others. This suggests that the conditional probabilities

$$\text{Prob}(|wt| < |w| \mid w), \quad t \in N_n$$

may not be equal for different non-minimal words $w \in F$, so the order in which Nielsen automorphisms are applied to an input w should depend on the word w itself. So the question we would like to address next is whether it is possible to find a dependence between a non-minimal word w and its length-reducing Nielsen automorphisms.

Here we claim that the K -means clustering algorithm allows one to discover some natural classes of non-minimal words. We show below that analysis of the corresponding cluster structures sheds some light on the relation between non-minimal words and their length-reducing automorphisms.

We apply the K -means algorithm to the set S_1 of non-minimal elements in free groups F_3, F_4, F_5 and which are represented by feature vectors $f(w) = f_{WG}(w)$ based on the Whitehead Graph of a word w (see Section 5.2.2).

To execute the K -means algorithm one has to define in advance the expected number of clusters K . Since we would like these clusters to be related to the set of Nielsen automorphisms N_n we put $K = |N_n|$.

To evaluate usefulness of the clustering we use the goodness measure R_{max} defined below. Let $\mathcal{C} \subset D$ be a cluster of the data set $D \subset F_n = F(X)$. For $t \in N_n$ define

$$R(t, \mathcal{C}) = \frac{|\{w \in \mathcal{C} \mid |wt| < |w|\}|}{|\mathcal{C}|}.$$

The number $R(t, \mathcal{C})$ shows how many elements in $\mathcal{C}$ are reducible by t . Now put

$$R_{max}(\mathcal{C}) = \max\{R(t, \mathcal{C}) \mid t \in N(X)\}$$

and denote by $t_{\mathcal{C}}$ a Nielsen automorphism $t \in N_n$ such that $R(t_{\mathcal{C}}, \mathcal{C}) = R_{max}(\mathcal{C})$. The number $R_{max}(\mathcal{C})$ shows how many elements in $\mathcal{C}$ can be reduced by a single automorphism, in this case by $t_{\mathcal{C}}$. We also define the average value of the goodness

measure

$$avg(R_{max}) = \frac{1}{K} \sum_{i=1}^K R_{max}(\mathcal{C}_i),$$

where K is the number of clusters.

The results of K -mean cluster analysis evaluated on sets of randomly generated non-minimal elements in free groups F_3, F_4, F_5 are given in Table 6.3. It shows that in each cluster there is one Nielsen automorphism satisfying that more than 70% of elements in the cluster can be reduced by this automorphism. In the free group F_3 , where the number of clusters is significantly smaller, the corresponding percentage is over 98%. Moreover, our experiments show that $t_{\mathcal{C}_i} \neq t_{\mathcal{C}_j}$ for $i \neq j$, see Figure 6.1. In other words there are no two distinct clusters such that one and the same Nielsen automorphism reduces most of the elements in both clusters.

	F_3	F_4	F_5
K	24	48	80
S_1	0.985	0.879	0.731
S_{10}	0.959	0.908	0.809
S_P	0.964	0.924	0.831

Table 6.3: Average values $avg(R_{max})$ of the goodness measure R_{max} for K -means clustering of test sets S_1, S_{10} and S_P . K is the number of clusters.

The discovered cluster structure gives rise to the following strategy in solving the Length Reduction Problem for a given word w . Let $\mu_1, \dots, \mu_K$ be the centers of clusters $\mathcal{C}_1, \dots, \mathcal{C}_K$ computed by the K -means procedure. We compute the distance $\|f(w) - \mu_i\|$ for each $i = 1, \dots, K$. Now we list the Nielsen automorphisms in N_n in the order $t_{i_1}, t_{i_2}, \dots, t_{i_K}$ with respect to the distances

$$\|f(w) - \mu_{i_1}\| \leq \|f(w) - \mu_{i_2}\| \leq \dots \leq \|f(w) - \mu_{i_K}\|.$$

To find a length reducing automorphism for a given word w we subsequently apply automorphisms from N_n in the prescribed order until we find an automorphism $t_i \in N_n$ which reduces the length of w . If such an automorphism does not exist we proceed with the remaining automorphisms from $\Omega_n - N_n$ as in the NF heuristic.

From the description of the K -means method we know that clusters are characterized by the center means of the feature vectors of elements in the same cluster. The observations above lead us to the following vaguely stated conjecture, which gives a model to describe behavior of non-minimal elements from F in terms of their feature vectors.

Conjecture 6.3.1. *The feature vectors of weights of the Whitehead Graphs of elements from F are separated into "compact" regions in the corresponding space. Each such region can be bounded by a hypersurface and corresponds to a particular Nielsen automorphism in a sense that all elements in the corresponding class can be reduced by that automorphism.*

6.4 Improvement on the clustering

Experiments with K -means clustering algorithm show that clustering is a useful tool in solving the length reduction problem. Now, the goal is to make clustering more effective. The further analysis of the clusters suggests that to some extent they correspond to partitions of elements in F which can be reduced by one and only one Nielsen automorphism. To verify this conjecture we perform the following experiment.

Let $D \subset F_n = F(X)$ be a set of randomly generated non-minimal elements

independent from a test set $S \subset F_n$. For each automorphism $t \in N_n$ put

$$O_t = \{w \in D \mid |wr| < |w| \iff r = t, \forall r \in N_n\}$$

and define

$$\lambda_t = \frac{1}{|O_t|} \sum_{w \in O_t} f(w) \quad (6.4.1)$$

as the mean feature vector of the elements from D that can be reduced only by t and no other Nielsen automorphism.

We cluster elements from S based on the distance between the corresponding feature vector and centers λ_t :

$$\mathcal{C}_t = \{w \in S \mid \|f(w) - \lambda_t\| \leq \|f(w) - \lambda_r\|, \forall r \in N_n\}.$$

The results of evaluation of the clusters $\mathcal{C}_t$ in free groups F_3, F_4, F_5 are given in Table 6.4. One can see that the goodness measure is improved and is close to 1 in every case.

	F_3	F_4	F_5
K	24	48	80
S_1	0.998	0.993	0.991
S_{10}	0.997	0.985	0.974
S_P	0.995	0.988	0.987

Table 6.4: Average values $avg(R_{max})$ of the goodness measure R_{max} for the clustering based on the distance to the estimated centers λ_t .

Similar to the strategy based on the centers of the K -means clusters, we define a new search procedure DWR_C which employs a heuristic based on the distances to centers λ_t . We will refer to this heuristic as *Centroid* heuristic. Let w be a word and

$\langle \lambda_{t_1}, \dots, \lambda_{t_K} \rangle$ be the centers corresponding to each of the Nielsen automorphisms $t_i \in N_n$. Put $d(i) = \|f(w) - \lambda_{t_i}\|$ and construct a vector

$$\langle d(m_1), d(m_2), \dots, d(m_K) \rangle,$$

where

$$d(m_1) \leq d(m_2) \leq \dots \leq d(m_K).$$

To find a length reducing automorphism for a given word w , the algorithm DWR_C applies Whitehead automorphisms to w in the following order. First, Nielsen automorphisms $t_{m_1}, \dots, t_{m_K}$ are applied subsequently. If none of the Nielsen automorphisms reduces the length of w the algorithm DWR_C proceed with the remaining automorphisms $\Omega_n - N_n$ in some fixed order.

Based on the results of the cluster analysis from Table 6.4, we expect the algorithm DWR_C to reduce a non-minimal word w using very few elementary automorphisms on average.

6.5 Maximal weight automorphisms first

Now we would like to take a closer look at the edge weight distributions at the cluster centers. First, observe that every edge in the Whitehead graph WG , except for the ones which correspond to subwords of type x^2 , $x \in X^{\pm 1}$, will correspond to subwords, reducible by two particular Nielsen transformations. For example, edge connecting nodes a and b corresponds to subwords $(ab^{-1})^{\pm 1}$ both of which are reduced by automorphisms

$$(a \rightarrow ab, b \rightarrow b),$$

$$(a \rightarrow a, b \rightarrow ba).$$

In fact there is no other Nielsen transformation that will reduce the length of words $(ab^{-1})^{\pm 1}$.

To generalize, let $WG(w)$ be a Whitehead graph of a word w with the vertex set V and the set of edges E . Let $e = (x, y^{-1})$, $x, y^{-1} \in V$ be an edge in E . By construction e corresponds to subwords $s_e = (xy)^{\pm 1}$ of the word w . The only Nielsen automorphisms which reduce length of the subwords s_e are

$$\psi_e^x : x \rightarrow xy^{-1}, z \rightarrow z \quad \forall z \neq x$$

and

$$\psi_e^y : y \rightarrow x^{-1}y, z \rightarrow z \quad \forall z \neq y.$$

We will call automorphisms ψ_e^x, ψ_e^y the length reducing Nielsen automorphisms with respect to the edge $e = (x, y^{-1})$ and denote $\psi_e = \{\psi_e^x, \psi_e^y\}$.

Recall that the weight $\omega(e)$ of the edge e is the number of times subwords s_e occur in the word w . We will denote $e_{max}(w)$ the edge in the graph $WG(w)$ having the maximal weight, i.e.

$$\omega(e_{max}) = \max\{\omega(e) \mid \forall e \in E\}.$$

In Table 6.5 we show examples of the edge weight distribution in the cluster centers λ_{t_1} and λ_{t_2} , $t_1 = (a \rightarrow ab^{-1}, b \rightarrow b, c \rightarrow c)$, $t_2 = (a \rightarrow a, b \rightarrow a^{-1}b, c \rightarrow c)$, obtained using Centroid method (see Equation 6.4.1) for the free group F_3 with basis $X = \{a, b, c\}$.

One can see that in each case the edge e_{max} having the maximal weight was

$$e_{max} = (a, b^{-1}).$$

Observe, that the clusters' transformations t_1 and t_2 are the corresponding length

Edge	Subwords	Weight
(a, b^{-1})	$(ab)^{\pm 1}$	0.22
(b, c)	$(bc^{-1})^{\pm 1}$	0.11
(b, c^{-1})	$(bc)^{\pm 1}$	0.11
(b, a^{-1})	$(ba)^{\pm 1}$	0.11
(b, b^{-1})	$b^{\pm 2}$	0.09
(c, c^{-1})	$c^{\pm 2}$	0.05
(c, a^{-1})	$(ca)^{\pm 1}$	0.05
(a^{-1}, c^{-1})	$(ac)^{\pm 1}$	0.05
(b^{-1}, c^{-1})	$(bc)^{\pm 1}$	0.05
(c, b^{-1})	$(cb)^{\pm 1}$	0.05
(a^{-1}, b^{-1})	$(ab)^{\pm 1}$	0.05
(a, b)	$(ab^{-1})^{\pm 1}$	0.02
(a, c)	$(ac^{-1})^{\pm 1}$	0.01
(a, c^{-1})	$(ac)^{\pm 1}$	0.01
(a, a^{-1})	$a^{\pm 2}$	0.01

a) λ_{t_1}

Edge	Subwords	Weight
(a, b^{-1})	$(ab)^{\pm 1}$	0.22
(c, a^{-1})	$(ca)^{\pm 1}$	0.11
(a^{-1}, c^{-1})	$(a^{-1}c)^{\pm 1}$	0.11
(b, a^{-1})	$(ba)^{\pm 1}$	0.11
(a, a^{-1})	$a^{\pm 2}$	0.09
(b, c)	$(bc^{-1})^{\pm 1}$	0.05
(c, c^{-1})	$c^{\pm 2}$	0.05
(b, c^{-1})	$(bc)^{\pm 1}$	0.05
(a, c^{-1})	$(ac)^{\pm 1}$	0.05
(a, c)	$(ac^{-1})^{\pm 1}$	0.05
(a, b)	$(ab^{-1})^{\pm 1}$	0.05
(a^{-1}, b^{-1})	$(a^{-1}b)^{\pm 1}$	0.02
(c, b^{-1})	$(cb)^{\pm 1}$	0.01
(b, b^{-1})	$b^{\pm 2}$	0.01
(b^{-1}, c^{-1})	$(b^{-1}c)^{\pm 1}$	0.01

b) λ_{t_2}

Table 6.5: Edge weight distributions in the cluster centers λ_{t_1} and λ_{t_2} of a free group $F_3(X)$, $X = \{a, b, c\}$, where $t_1 = (a \rightarrow ab^{-1}, b \rightarrow b, c \rightarrow c)$ and $t_2 = (a \rightarrow a, b \rightarrow a^{-1}b, c \rightarrow c)$. Rows are sorted by the edge weights.

reducing Nielsen transformation for the edge (a, b^{-1}) . In other words

$$t_1 = \psi_{e_{max}}^a \text{ and } t_2 = \psi_{e_{max}}^b.$$

In fact the same phenomenon was observed for all clusters in free groups F_3 , F_4 , and F_5 . Let $\mathcal{C}_t$ be a cluster of a test set S , then for all $t \in N_n$, $n = 3, 4, 5$

$$t \in \psi_{e_{max}},$$

where e_{max} is the edge having the maximal weight in the cluster center λ_t . It suggests that at least in the case of free groups F_3 , F_4 , F_5 one can try to find a length-reducing automorphism for given word w by taking the length-reducing Nielsen automorphisms of the highest weight edge in the Whitehead graph $WG(w)$.

To evaluate the goodness of the heuristic based on the edge weight in the Whitehead graph we compute the fraction of elements in the test sets S_1 , S_{10} and S_P , reducible by the Nielsen automorphisms corresponding to the maximal weight edge. The corresponding goodness measure, evaluated on a set S , is given by

$$\mathcal{G}_{MAX} = \frac{|\{w \in S \mid \exists t \in \psi_{e_{max}(w)}, \text{ s.t. } |wt| < |w|\}|}{|S|}.$$

	F_3	F_4	F_5
S_1	0.991	0.986	0.986
S_{10}	0.980	0.959	0.947
S_P	0.968	0.974	0.979

Table 6.6: Values of the goodness measure $\mathcal{G}_{MAX}$ for sets of non-minimal elements in free groups F_3 , F_4 and F_5 .

Values of the goodness measure $\mathcal{G}_{MAX}$ for test sets in free groups F_3 , F_4 and F_5 are given in Table 6.6. It shows, that heuristic is surprisingly effective. Nevertheless,

centroid based method still yields better results. Note that $\mathcal{G}_{MAX}$ measures success of applying two automorphisms corresponding to the maximal weight edge, where the centroid based method was evaluated by the success rate of only one automorphism which corresponds to the closest center.

This observation gives rise to a new search procedure which we denote by DWR_{MAX} . Let w be a word and $WG(w) = (V, E)$ be the corresponding Whitehead graph. Denote by E' the set of edges which do not correspond to the subwords of type $x^{\pm 2}$, $x \in X$

$$E' = \{e \in E \mid e \neq (v, v^{-1}), v \in V\}.$$

It has been shown above, that for each edge e from E' there exists two unique length reducing automorphisms. Note that $2|E'| = |N_n|$, where N_n is the set of Nielsen automorphisms for free group F_n .

We can order Nielsen automorphisms $\psi_{e_i} \subset N_n$:

$$< \psi_{e_1}, \psi_{e_2}, \dots, \psi_{e_{|E'|}} > \tag{6.5.1}$$

such that edges $e_1, \dots, e_{|E'|}$ are chosen according to the decreasing order of the values of the corresponding weights

$$\omega(e_1) \geq \omega(e_2) \geq \dots \geq \omega(e_{|E'|}).$$

Note that ψ_e is not a single automorphism, but a pair of Nielsen length reducing automorphisms with respect to the edge e . Here we do not give any preference in ordering automorphisms in ψ_e .

To find a length-reducing automorphism for w the procedure DWR_{MAX} first applies Nielsen automorphisms in the order given by (6.5.1). If neither of the Nielsen

automorphisms reduces the length of w , DWR_{MAX} proceeds with the remaining automorphisms from $\Omega_n - N_n$.

6.6 Comparison of the strategies

6.6.1 Estimates of the expected performance

Here we try to estimate the expected number of steps required to solve the length-reduction problem by each of the heuristic algorithms DWR_{NF} , DWR_C , and DWR_{MAX} .

Note that the algorithms differ in the way they choose the order of Nielsen automorphisms in the search procedure. All of the algorithms above proceed in the same way after no Nielsen length-reducing automorphism has been found for the input word. We can divide the Length Reduction Procedure of the heuristic algorithms in two parts:

1. Ordering and application of Nielsen automorphisms. We will refer to the implementation of this stage in an algorithm $\mathcal{A}$ as to $ON_{\mathcal{A}}$. By $C_{\mathcal{A}}$ we denote the time complexity of this stage in an algorithm $\text{DWR}_{\mathcal{A}}$;
2. Application of the rest $\Omega_n - N_n$ Whitehead automorphisms if no Nielsen length-reducing automorphism exists. The time complexity of the second stage is denoted by C_{Ω} .

The expected time complexity of an algorithm $\text{DWR}_{\mathcal{A}}$ to find a length reducing automorphism for a given input $w \in F_n$ of length l is

$$cl(P_l E[C_{\mathcal{A}}] + (1 - P_l) E[C_{\Omega}]),$$

where $E[C_{\mathcal{A}}]$ and $E[C_{\Omega}]$ are the expected time complexities of parts 1 and 2 of the algorithm $\text{DWR}_{\mathcal{A}}$, $P_l = |NS_{NMin}^l|/|S_{NMin}^l|$ (see Section 6.2) and c is some constant which does not depend on n or l .

Since part 2 is identical for all three algorithms DWR_{NF} , DWR_C and DWR_{MAX} , we will concentrate on the analysis of the first part. The time complexity is estimated in the number of elementary steps, where by elementary step we mean one application of a Whitehead automorphism to a given word.

Expected Performance of Nielsen-First strategy

To find a length-reducing automorphism, Nielsen-First strategy simply applies automorphisms from the set N_n first. The expected number of elementary steps to find a Nielsen length-reducing automorphism (if it exists) using NF strategy is

$$E[C_{NF}] = \sum_{i=1}^{|N_n|} P_{NF}(i)i,$$

where $P_{NF}(i)$ is the probability of finding a length-reducing Nielsen automorphism using NF strategy in i steps. Unfortunately, we do not know the real probabilities P_{NF} . We can use empirical estimates $\hat{P}_{NF}$ of the probabilities P_{NF} to obtain the approximate value of $E[C_{NF}]$.

The estimates $\hat{P}_{NF}(i)$ can be defined as fractions of elements from a test set S , which are reduced in i elementary steps, given a fixed random order of automorphisms in N . In general, let

$$N^o = \langle t_1, \dots, t_{|N|} \rangle$$

be an ordered set of Nielsen automorphisms, then the probability estimates with

respect to the ordering o are defined

$$\hat{P}_o(i) = \frac{|\{w \in S \mid |t_i w| < |w|, |t_j w| \geq |w|, j < i, t_i, t_j \in N^o\}|}{|S|},$$

where $i = 1, \dots, |N_n|$. Values of $\hat{P}_{NF}$, computed for free groups F_3 , F_4 and F_5 are shown in Figures 6.2 and 6.3.

To get a better understanding of how effective this method is, we compute the 99th percentile of the number of Nielsen automorphisms required to reduce a given non-minimal word from a test set. Let S be a test set and $\hat{P}_o(i)$ be the corresponding probability estimates. Define a 99th percentile

$$PCT_{99} = \min \left\{ k \mid \sum_{i=1}^k \hat{P}_o(i) \geq 0.99 \right\}.$$

In Table 6.7a we give values of PCT_{99} computed for Nielsen First heuristic. One can see that for test sets S_1 the number of automorphisms which guarantee a successful reduction of 99% of the elements in S_1 is very high. It ranges from 88% of Nielsen automorphisms in the case of the free group F_5 to 96% in the case of F_3 . For test sets S_{10} the number of automorphisms decreases, but it is still about 63% in the best case.

Expected Performance of Centroid heuristic

To implement the heuristic, based on the cluster centers, first, we need to construct the Whitehead graph for the input word $w \in F_n$. This requires one scanning of w and storing the weights of all edges to obtain the feature vector. The time complexity of this stage is

$$C_{WG} = |w| + 2n(2n - 1) \sim O(|w| + n^2).$$

	F_3	F_4	F_5
$ N_n $	24	48	80
S_1	23	45	70
S_{10}	21	37	50

a) Nielsen First heuristic;

	F_3	F_4	F_5
$ N_n $	24	48	80
S_1	1	1	1
S_{10}	1	2	3

b) Centroid heuristic;

	F_3	F_4	F_5
$ N_n $	24	48	80
S_1	2	3	4
S_{10}	6	7	6

c) Maximal Weight heuristic;

Table 6.7: Values of PCT_{99} computed for different heuristics applied to test sets S_1 and S_{10} in free groups F_3 , F_4 and F_5 .

Next, we have to compute distances to each of the cluster centers and sort them. The cost of this operation is

$$C_d = c2n(2n - 1) \cdot |N_n| + 2n \log 2n \sim O(|w| + n^4),$$

where $2n(2n - 1)$ is the size of the feature vector, $|N_n| = 4n(n - 1)$ and $2n \log 2n$ is the cost of sorting weights.

Now the expected time complexity of ON_C in the Centroid based heuristic is

$$E[C_C] = C_{WG} + C_C + c|w| \sum_{i=1}^{|N_n|} P_C(i)i,$$

where c is a constant.

The probability estimates $\hat{P}_C$ with respect to the ordering spawned by the Centroid based heuristic and the corresponding 99th percentiles are given in Figures 6.4, 6.5 and Table 6.7b. It is amazing that using this strategy we need only one automorphism to be able to reduce 99% of non-minimal elements from S_1 . This shows that Centroid based strategy is very efficient on most inputs.

Expected Performance of Maximal Weight heuristic

In the implementation of ON_{MAX} we again construct the Whitehead graph of a word w and then sort the corresponding weights. The cost is

$$c(|w| + 2n(2n - 1) + 2n \log 2n) \sim O(|w| + n^2).$$

Therefore, the expected complexity

$$E[C_{MAX}] = c_1(|w| + 2n(2n - 1) + 2n \log 2n) + c_2|w| \sum_{i=1}^{|N_n|} P_{MAX}(i)i.$$

The estimates for probabilities P_{MAX} are given in Figures 6.6 and 6.7. From Table 6.7c we can see that using Maximal Weight heuristic we need very few automorphisms to reduce most of non-minimal elements.

6.6.2 Experimental results

In this section we describe experiments designed to compare the performance of DWR implemented with different search strategies. We compare four variations of the algorithm. DWR_R is the variation of DWR, where a random order of the elements from Ω_n is used when searching for length reducing automorphisms. DWR_{NF} and DWR_C correspond to the implementations with Nielsen First and Centroid based

heuristics respectively. The algorithm DWR_{MAX} employs strategy which applies automorphisms corresponding to the largest edge weights of the Whitehead Graph. The algorithms were compared on randomly generated sets of primitive elements in free groups F_3 , F_4 , and F_5 , described in Table 6.1. Primitive elements are suited very well for performance comparison of the Whitehead type algorithms because of their high Whitehead complexity.

Let $\mathcal{A}$ be one of the variations DWR_R , DWR_{NF} , DWR_C , and DWR_{MAX} of the Whitehead Reduction Algorithm. By an elementary step of the algorithm $\mathcal{A}$ we mean one application of a Whitehead automorphism to a given word. Below we evaluate the performance of $\mathcal{A}$ with respect to the number of elementary steps that are required by $\mathcal{A}$ to execute a particular routine.

Let $N_{total} = N_{total}(\mathcal{A}, S_n)$ be the average of the total number of elementary steps required by $\mathcal{A}$ to reduce a given primitive element $w \in S_n$ to a generator.

By $N_{red} = N_{red}(\mathcal{A}, S_n)$ we denote the average number of elementary length-reducing steps required by $\mathcal{A}$ to reduce a given primitive element $w \in S_n$ to a generator, so N_{red} is the average number of "productive" steps performed by $\mathcal{A}$. It follows that if $t_1, \dots, t_l$ are all the length reducing automorphisms found by $\mathcal{A}$ when executing its routine on an input $w \in S_n$ then $|wt_1 \dots t_l| = 1$ and the average value of l is equal to N_{red} .

Finally, denote by $N_{LRP} = N_{LRP}(\mathcal{A}, S_n)$ the average number of elementary steps required by $\mathcal{A}$ to find a length-reducing automorphism for a given non-minimal input w .

In Table 6.8 we present results of our experiments on performance of the algorithms DWR_R , DWR_{NF} , DWR_C and DWR_{MAX} on the test sets S_n , $n = 3, 4, 5$. The

algorithms compare as expected. The algorithms DWR_C and DWR_{MAX} perform very efficiently with the numbers N_{LRP} and N_{red} being small. Algorithm DWR_C based on the centroid approach shows best over all performance. The growth of the numbers N_{LRP} with the rank could be explained by occasional occurrence of non-minimal words that cannot be reduced by Nielsen automorphisms. In this event the algorithm tries Whitehead automorphisms from $\Omega_n - N_n$ the number of which growth exponentially with the rank.

Strategy	N_{total}	N_{red}	N_{LRP}
DWR_C	19.9	18.4	1.1
DWR_{MAX}	47.1	23.9	1.9
DWR_{NF}	207.8	28.2	7.37
DWR_R	374.8	29.8	12.6

a) F_3 ;

Strategy	N_{total}	N_{red}	N_{LRP}
DWR_C	58.8	34.1	1.4
DWR_{MAX}	152.8	42.5	3.0
DWR_{NF}	1052.6	56.2	18.7
DWR_R	2610.4	58.8	44.4

b) F_4 ;

Strategy	N_{total}	N_{red}	N_{LRP}
DWR_C	162.0	50.9	2.4
DWR_{MAX}	342.2	58.8	4.5
DWR_{NF}	2307.6	75.4	30.6
DWR_R	15939.6	78.8	202.0

c) F_5 .

Table 6.8: Results of experiments with sets of primitive elements in free groups F_3, F_4 and F_5 . Counts are averaged over all inputs.

In Figures 6.8 we present graphs showing the dependence of the number of elementary steps required by a particular algorithm to find a length-reducing automorphism with respect to the length of the input words. It shows that these numbers seem to depend only on the rank of the group, not on the length of the word. Indeed, in Table 6.9 we give the correlation coefficients between the number of elementary steps and the length of the input words. The coefficients are negative in all cases which indicates that the values of N_{LRP} do not increase when the words' length increases.

Strategy	F_3	F_4	F_5
WRA_C	-0.008	-0.001	-0.006
WRA_{MAX}	-0.024	-0.023	-0.038
WRA_{NF}	-0.009	-0.022	-0.022
WRA_R	-0.038	-0.014	-0.035

Table 6.9: Correlation coefficients between a word's length and values of N_{LRP} . Negative coefficients indicate that N_{LRP} does not increase when length increases.

Conclusion 6.6.1. *The experimental results show that using appropriate heuristics in the algorithm DWR, one can significantly reduce the complexity of the Whitehead minimization problem on most inputs with respect to the group rank. Suggested heuristics reduce the average number of Whitehead automorphisms required to find a length-reducing automorphism for a given word. The performance of heuristic algorithms tested on the sets of randomly generated primitive elements shows robust behavior and does not deteriorate when the length of the input words increases.*

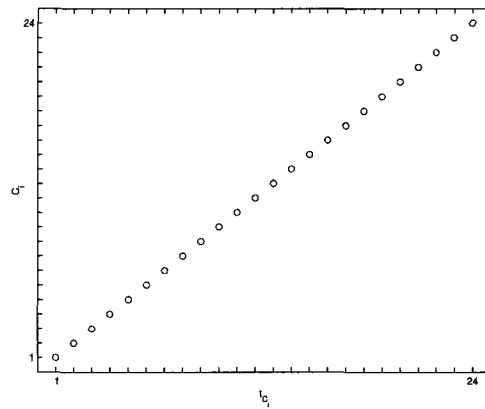
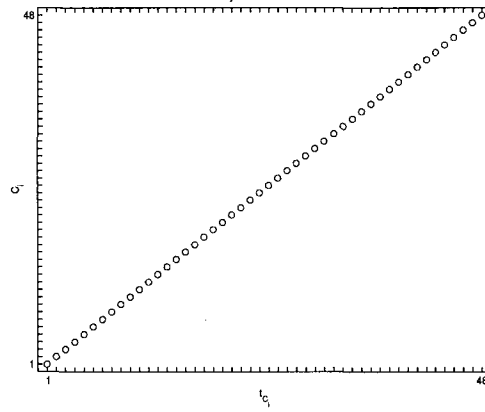
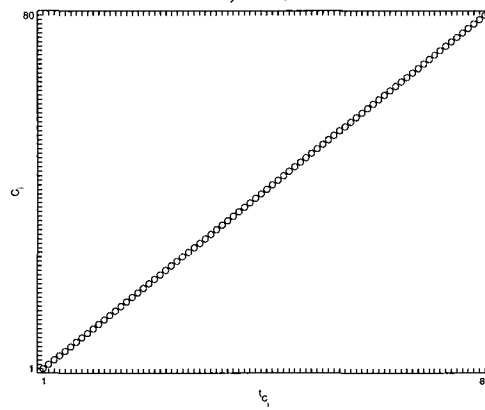
a) F_3 ;b) F_4 ;c) F_5 .

Figure 6.1: Each cluster C_i in free groups F_3 , F_4 and F_5 is plotted against the corresponding transformation t_{C_i} .

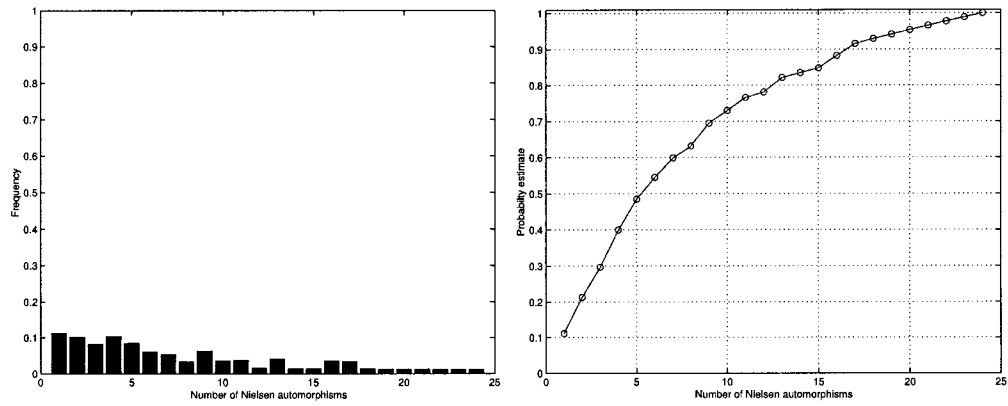
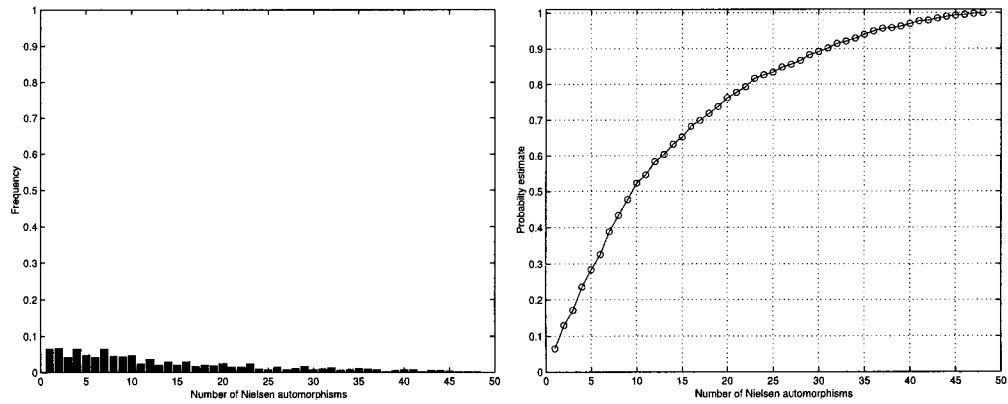
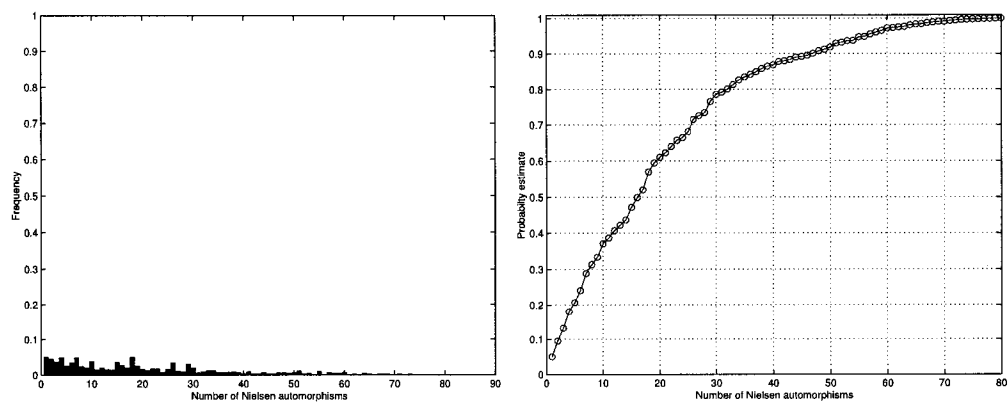
a) F_3 ;a) F_4 ;a) F_5 ;

Figure 6.2: Values of estimates $\hat{P}_{NF}$ obtained from the test set S_1 in free groups F_3 , F_4 and F_5 .

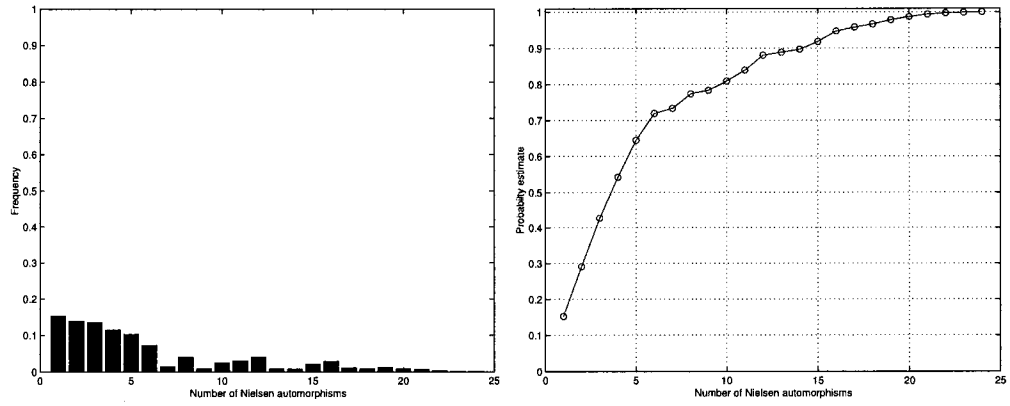
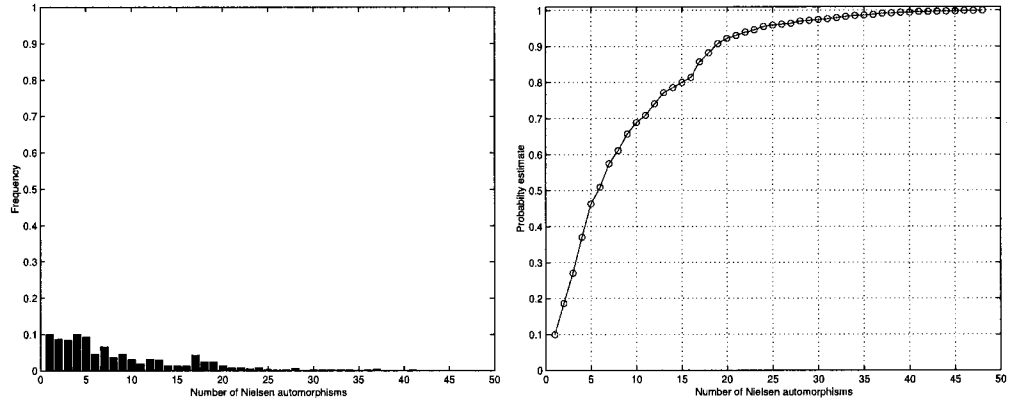
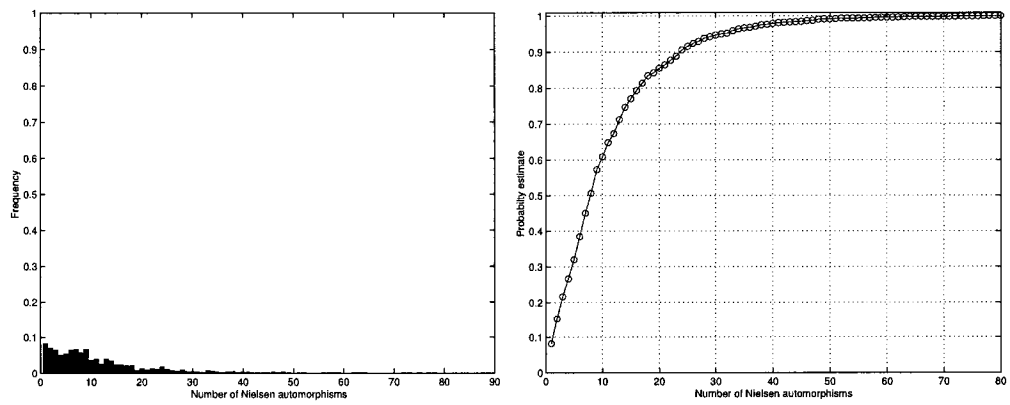
a) F_3 ;a) F_4 ;a) F_5 ;

Figure 6.3: Values of estimates $\hat{P}_{NF}$ obtained from the test set S_{10} in free groups F_3 , F_4 and F_5 .

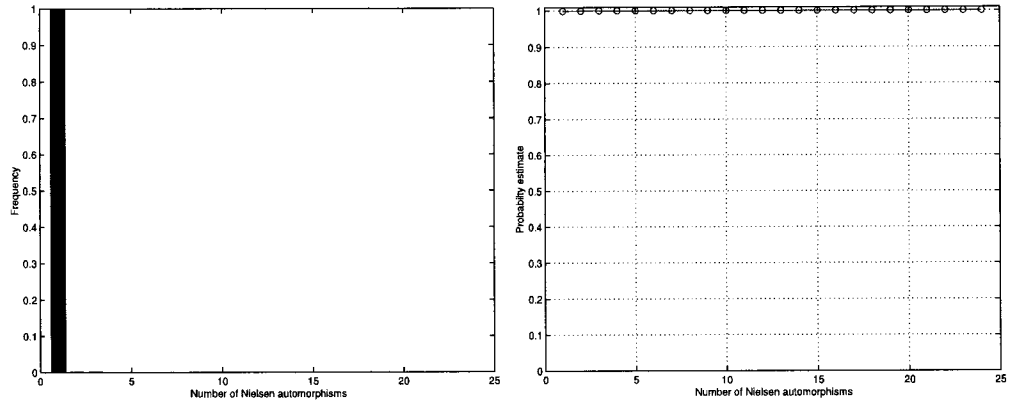
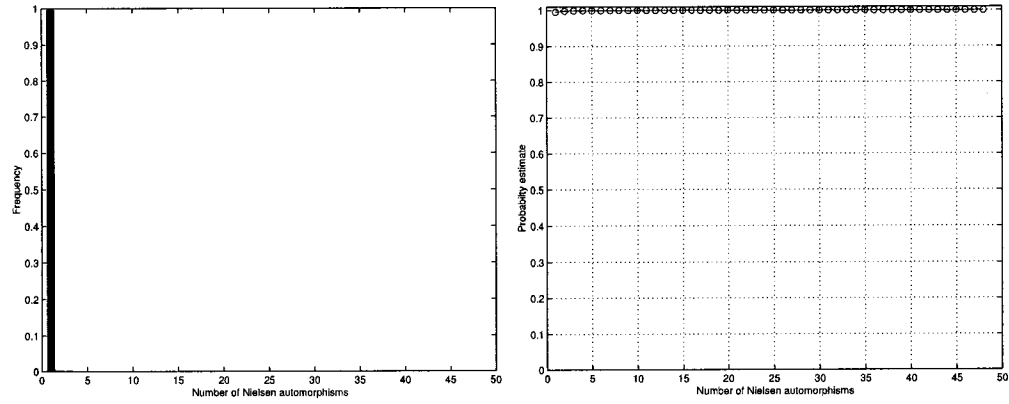
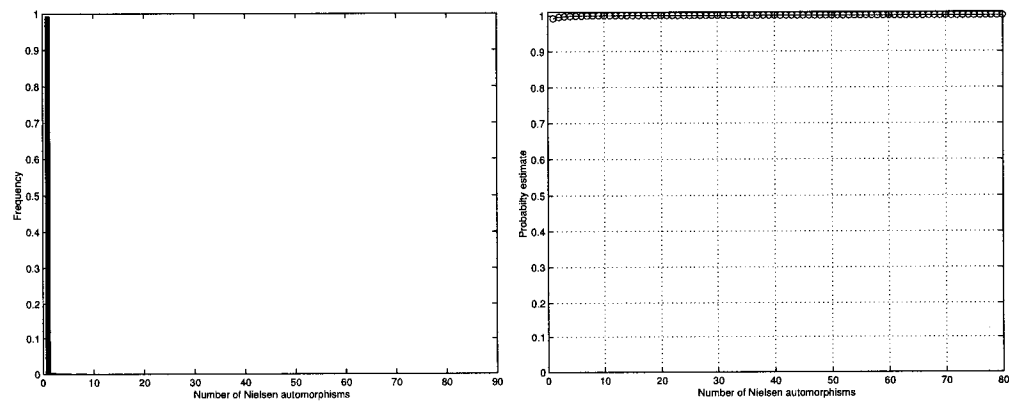
a) F_3 ;a) F_4 ;a) F_5 ;

Figure 6.4: Values of estimates $\hat{P}_C$ obtained from the test set S_1 in free groups F_3 , F_4 and F_5 .

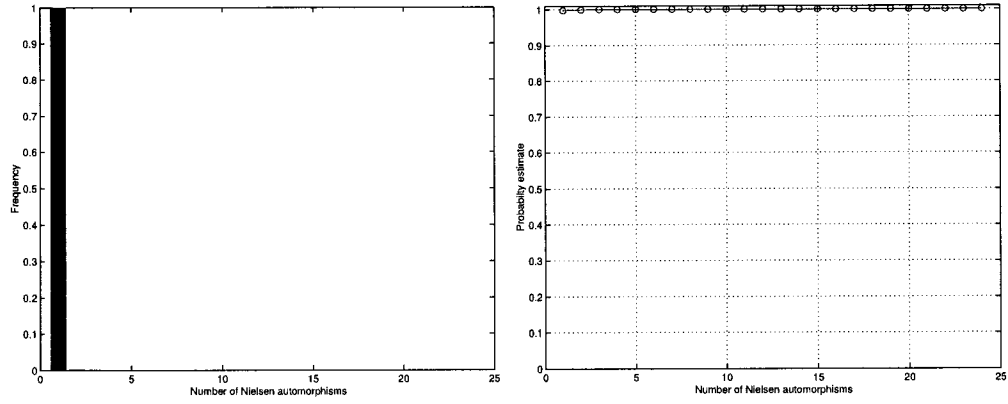
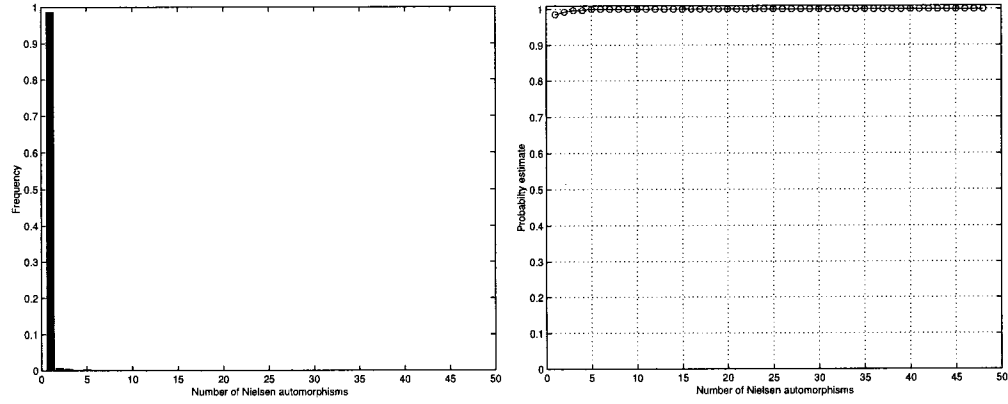
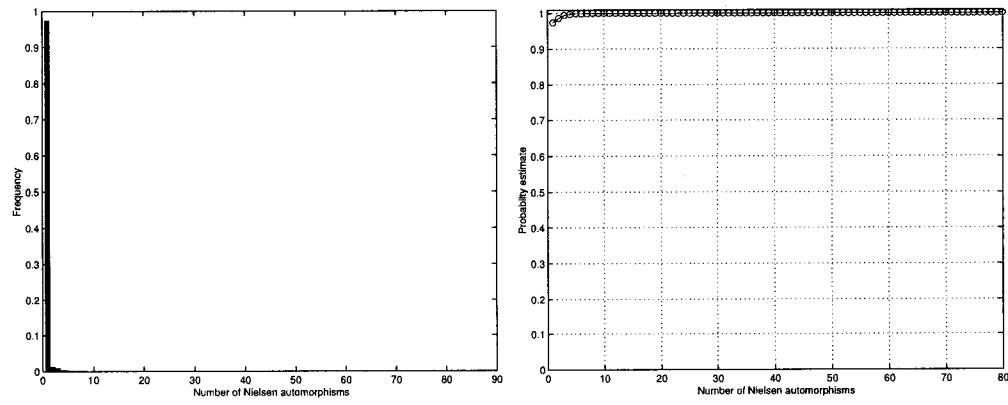
a) F_3 ;a) F_4 ;a) F_5 ;

Figure 6.5: Values of estimates $\hat{P}_C$ obtained from the test set S_{10} in free groups F_3 , F_4 and F_5 .

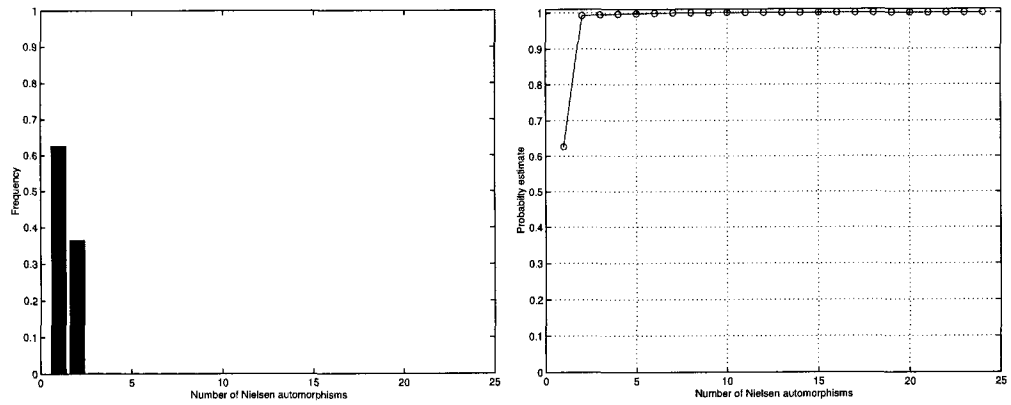
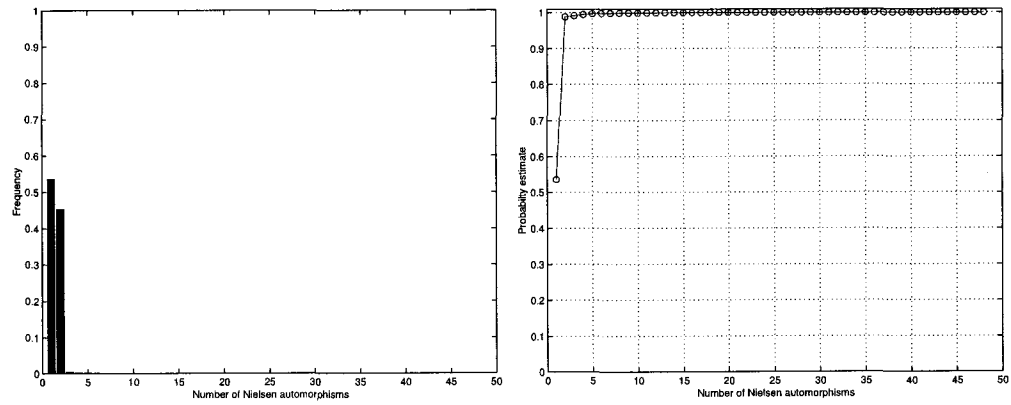
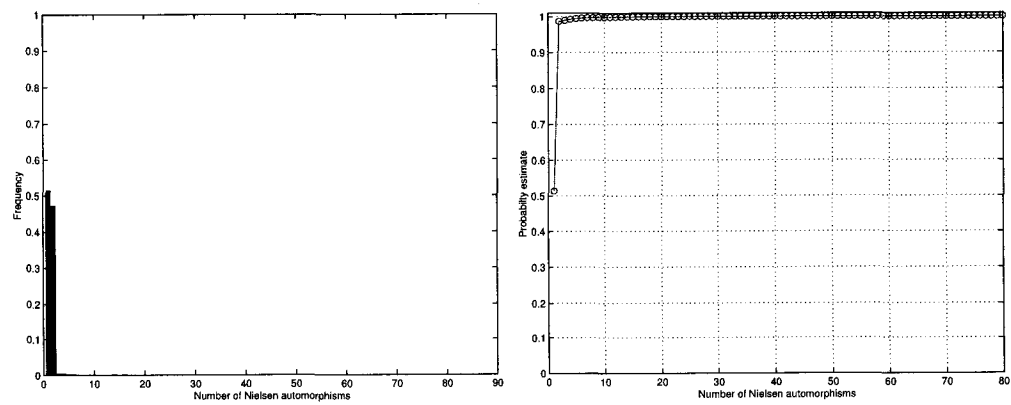
a) F_3 ;a) F_4 ;a) F_5 ;

Figure 6.6: Values of estimates $\hat{P}_{MAX}$ obtained from the test set S_1 in free groups F_3 , F_4 and F_5 .

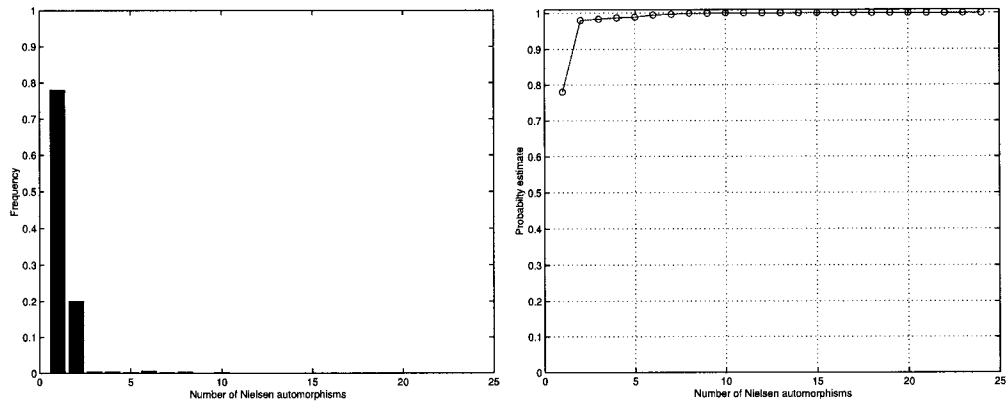
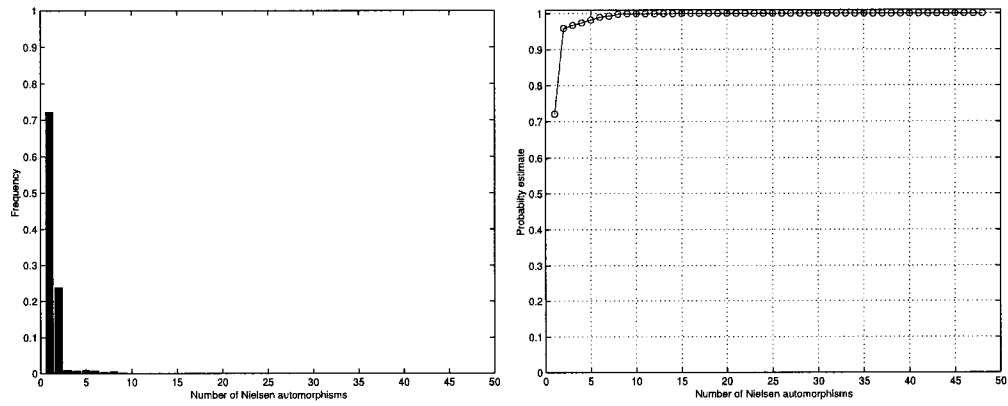
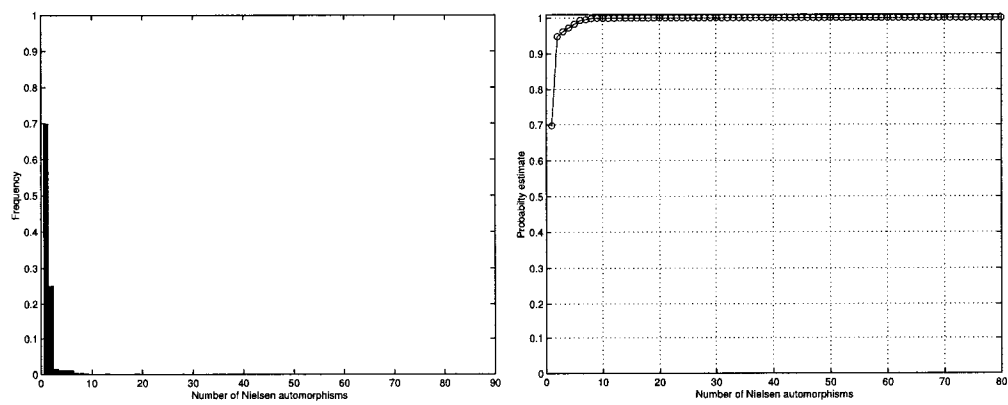
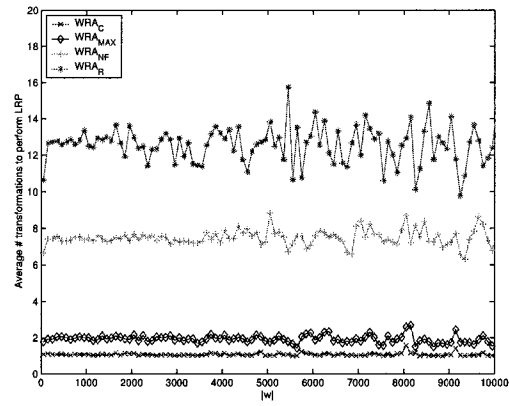
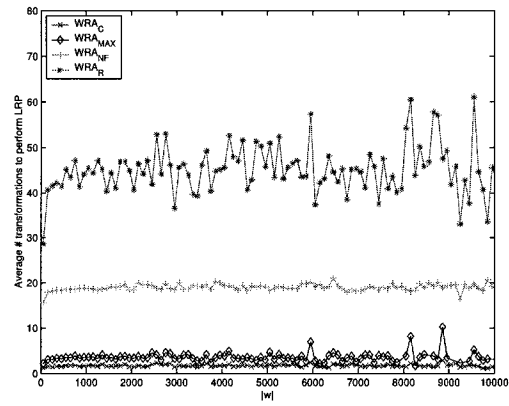
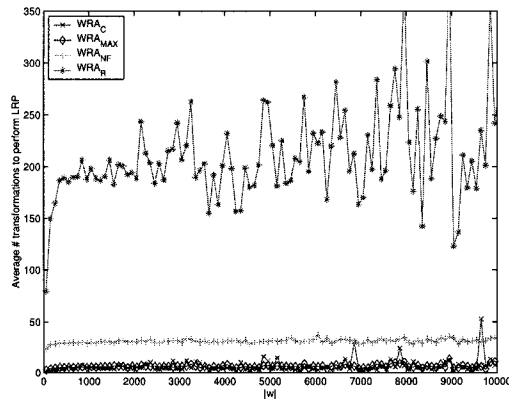
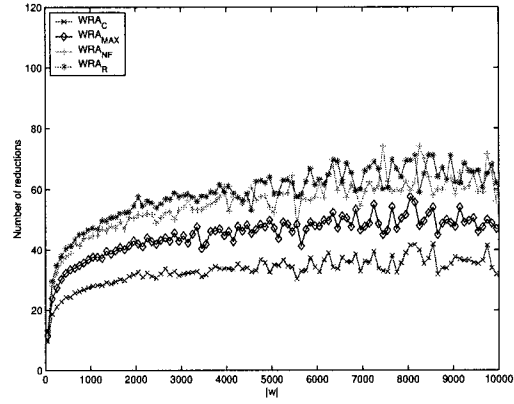
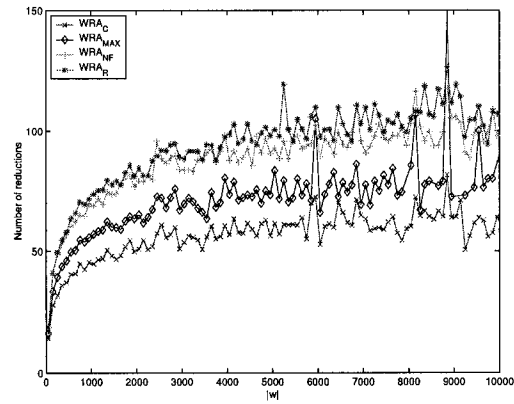
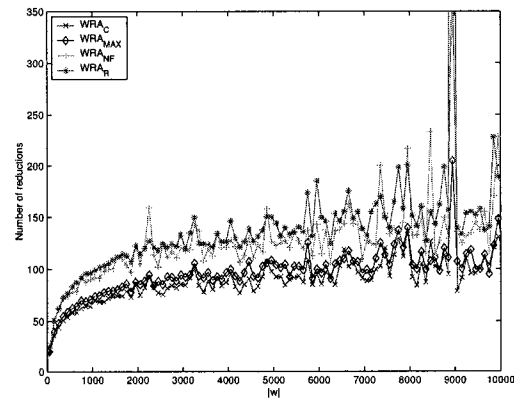
a) F_3 ;a) F_4 ;a) F_5 ;

Figure 6.7: Values of estimates $\hat{P}_{MAX}$ obtained from the test set S_{10} in free groups F_3 , F_4 and F_5 .

a) F_3 ;b) F_4 ;c) F_5 .Figure 6.8: Values of N_{LRP} for DWR_R , DWR_{NF} , DWR_C and DWR_{MAX} .

a) F_3 ;b) F_4 ;c) F_5 .Figure 6.9: Values of N_{red} for DWR_R , DWR_{NF} , DWR_C and DWR_{MAX} .

Chapter 7

A Hybrid Search Algorithm for the Whitehead Minimization Problem

7.1 Introduction

In this chapter we introduce a new algorithm for solving Whitehead minimization problem. It is a *hybrid* algorithm in a sense that it employs several stochastic as well as deterministic procedures during the search.

We combine stochastic search algorithm SWR and the probabilistic classification system WMIN described in Chapters 4 and 5 with a variation of heuristic search procedures, described in the previous chapter, to construct a Hybrid Deterministic Whitehead Reduction (HDWR) algorithm which solves the Length Reduction Problem in a constant number of steps (in terms of the group rank) on most input words from a free group.

The resulting algorithm is deterministic and still requires exponential number of steps to prove that a word is minimal. We present a fast probabilistic algorithm HPWR, which is a slight modification of HDWR. Algorithm HPWR is very robust and extremely fast on most input words, including words in free groups of large ranks.

The algorithms HDWR and its probabilistic version HPWR are described in the next section. We give experimental results evaluating the performance of the two algorithms in Section 7.3. Comparison with the standard deterministic procedure is also presented in the last section of the chapter.

7.2 Description of the Hybrid algorithms

7.2.1 HDWR

The flowchart diagram of the deterministic hybrid procedure HDWR is given in Figure 7.1. The algorithm contains two major parts. The first part consists of a number of so-called *fast checks* - linear or polynomial procedures which can solve length reduction problem on some inputs. In fact, the fast checks used in HDWR are expected to reduce most of non-minimal elements in a free group. The problem is that there are non-minimal words which cannot be reduced by fast procedures. Using fast checks alone one cannot decide whether an input word is minimal or not. We need to provide a termination condition of the algorithm. This task is solved by the second part of the algorithm which is a version of a standard deterministic algorithm DWR. Note that in most cases, the computationally ineffective procedure DWR is expected to be executed only on minimal elements.

HDWR is an iterative procedure. On each iteration the length reduction problem is solved for the word w_c which is an automorphic image of the minimal length of the input word w found so far. The algorithm terminates when there are no reductions possible and the current word w_c is returned as a minimal word w_{min} .

The first step in the algorithm is the classification procedure WMIN (step S2

in the diagram) which decides whether a current word is minimal or not. Even though reduction procedures used as fast checks do not require a lot of computational resources, step S2 helps to avoid superfluous computations by distinguishing minimal elements on the first stage of the algorithm.

Fast reduction procedures are based on the search heuristics described in the Chapter 6. From the analysis in Section 6.6 we know that Maximal Weight heuristic has the smallest time complexity on average. Note that more then 90% of non-minimal elements are expected to be reduced using one of the two automorphisms corresponding to the maximal weight edge of $WG(w_c)$.

The first fast check is based on the Maximal Weight heuristic. Let $e_{max}(w_c)$ be the maximal edge in $WG(w_c)$ and $\psi_{e_{max}}$ be the set of two length-reducing automorphisms with respect to the edge $e_{max}(w_c)$. In the step S3 of the algorithm HDWR we apply automorphisms $\psi_{e_{max}}$ to w_c . The maximal number of steps, required to perform the fast check at step S3 is

$$O(n^2 + |w_c|).$$

These steps include the construction of the feature vector $f_{WG}(w_c)$ and the application of the automorphisms $\psi_{e_{max}}$. Following the observations from Section 6.6 we expect most of non-minimal elements to be reduced at the step S3. If the word w_c has been reduced by one of the two automorphisms, say $\psi'_{e_{max}}$, we return to step S1, substitute w_c with $\psi'_{e_{max}}(w_c)$ and start a new iteration. If the check S3 fails, we continue the reduction process with the next step.

In the step S4 we use Centroid heuristic described in Section 6.4. Let $\psi_{e_{max}}$ be the set of two automorphisms applied at the step S3 and N_n be the Nielsen set as

usual. We order automorphisms

$$\langle \varphi_1, \dots, \varphi_K \rangle, \varphi_i \in N_n - \psi_{e_{max}}, \quad (7.2.1)$$

such that

$$d_1 \leq d_2 \leq \dots \leq d_K,$$

where $d_i = \|f(w_c) - \lambda_{\varphi_i}\|$ is the distance from the feature vector of the word w_c to the cluster center corresponding to the automorphism φ_i .

We apply automorphisms $\varphi_1, \dots, \varphi_K$ in the order, given by (7.2.1) to the word w_c . If one of the automorphisms has reduced the length of w_c , we stop and start a new iteration from the Step S1 with the new, reduced word w_c . The maximal number of steps required to execute S4 is

$$O(n^4 + n^2|w_c|).$$

At the step S5 we already know that the word w_c does not have Nielsen length-reducing automorphisms. The suggested strategy at this point is to try to reduce w_c by executing SWR for a predefined number of generations. If SWR fails to find a length-reducing automorphism, we continue with the algorithm DWR. The maximal number of generations of the stochastic algorithm can be estimated using values $E(r)$ from Table 4.4. A very conservative bound has been given in the termination condition T3 of the algorithm. Note, that since algorithm SWR performs better than DWR only in groups with relatively large ranks, it might happen that performance improves if we omit step S5 when the rank of a free group is small.

The maximal time complexity to find a length-reducing automorphism for w_c using HDWR is still

$$O(2^n|w_c|).$$

However, following the results from Section 6.6, we expect the reduction process to be extremely efficient for most non-minimal words. Unfortunately, the worst case behavior of the algorithm occurs when the current word w_c becomes a minimal. Except for some trivial cases (when a minimal word is a generator for example) it seems no algorithm is known to avoid the time-consuming computation. In the next section we introduce a probabilistic algorithm which addresses this problem.

7.2.2 HPWR

Words of both types, minimal and non-minimal, may cause an exponential blowout in the algorithm HDWR. However, the non-minimal words do not seem to be a major problem since we have shown that most of them can be reduced by one of the Nielsen automorphisms.

The bottleneck in solving the length reduction problem is the lack of a fast algorithm to decide whether a word is minimal or not. In fact, the only known deterministic solution is the algorithm DWR itself. Recall that the worst case of the algorithm occurs when the input word is already minimal. In this case all of the Whitehead automorphisms have to be applied to the word before the decision that the word is minimal is made.

In this section we introduce a Hybrid Probabilistic Whitehead Reduction algorithm HPWR for solving Whitehead's minimization problem. In HPWR the decision on whether a word is minimal or not is made using probabilistic classification system WMIN. This allows one to avoid the exponential blowout for the price of a small classification error.

We construct HPWR from HDWR by, first, removing the last step (step S6) from

the algorithm (see Figure 7.2). Note the increased role of the stochastic algorithm SWR. This is the only method in HPWR which is capable of reducing non-minimal elements which do not have Nielsen length-reducing automorphisms.

Secondly, we move the classification step behind the fast reduction procedures. To explain this modification we would like to return to the discussion of the roles played by the two types of classification errors of the classifier WMIN in the view of its new application.

Recall that the two errors are: the false positive error which is an error of classifying a non-minimal element as minimal and the false negative error is an error of classifying a minimal element as a non-minimal element. Now observe, that once the classifier WMIN decides that the word w_c is minimal, algorithm HPWR terminates and returns w_c as the result. There is no backtracking or additional checking performed after the decision is made. This means that if a non-minimal word w_c is classified as minimal the algorithm will produce an incorrect result. On the contrary, when a minimal word is misclassified as non-minimal, the cost of such error is the number of extra computational steps performed by the algorithm in order to reduce a non-reducible word. What is important is that the algorithm still produces a correct result.

Let α be the probability of committing the false positive error by WMIN. Now assume, that during the reduction process classifier WMIN was called k times to decide whether an element w_c is minimal or not. The probability that algorithm terminates with a correct answer is $(1 - \alpha)^k$. This shows that the probability of giving an incorrect answer grows rapidly with the number of times the minimality decision is made.

Note that most of the reductions are expected to be done by fast check procedures in steps S2 and S3. Moving classification step behind the fast checks allows us to reduce the error of producing an incorrect answer while still maintaining a small computational cost on average.

The arguments above show that the false positive error of the classifier at the step S2 of the algorithm HPWR has crucial importance. It is necessary to keep rate of the false positive error as minimal as possible in order for the algorithm to perform correctly. One can control the rates of the false positive/negative errors by adjusting the prior probabilities $Pr(P(w) = 1)$ and $Pr(P(w) = 0)$ of a randomly chosen word w being minimal or non-minimal (see Section 5.2.3). Given $Pr(P(w) = 0)$ a higher values will cause the Bayes decision rule (5.2.3) to decide that a word is non-minimal more often and, therefore, to decrease the rate of the false positive error.

7.3 Evaluation

In this section we evaluate the algorithms HDWR and HPWR and compare their performance with the performance of the algorithm DWR.

For the evaluation measures we use the average number of elementary steps N_s required by an algorithm to find a minimal element for a given word and the average number of elementary length-reducing steps N_{red} required by an algorithm to reduce a given element to a minimal one (see Section 6.6.2). In addition we record the CPU time $T(w)$ spent by an algorithm to produce a solution for a particular word w . Since HPWR is a probabilistic algorithm there exists a possibility of producing an incorrect solution. We measure the error of a probabilistic Whitehead reduction algorithm $\mathcal{A}$ by computing the fraction of elements for which $\mathcal{A}$ failed to return a minimal element.

Let $Sol_{\mathcal{A}}(w) \in F_n$ be a solution produced by an algorithm $\mathcal{A}$. If result is correct, then $|Sol_{\mathcal{A}}(w)| = |w_{min}|$. Now, let $S \subset F_n$ be a test set. The error rate of $\mathcal{A}$ with respect to the test set S

$$E(\mathcal{A}) = \frac{|\{w \in S \mid |Sol_{\mathcal{A}}(w)| > |w_{min}|\}|}{|S|}.$$

We evaluate algorithms on the test datasets described in Table 6.1. In all experiments values of the prior probabilities in the classifier were taken to be equal. We used termination condition T3 to stop SWR with the maximal number of generations equal to $5E(r)$.

First, we experiment in groups of smaller ranks. For elements in free groups F_3 , F_4 and F_5 algorithm DWR can decide in a practically acceptable amount of time on whether an element is minimal or not. This allows us to obtain the true values of lengths of minimal elements for each of the input words and access the error rate of probabilistic algorithms. Results are presented in Tables 7.1 - 7.3, where

$$T_{avg} = \frac{1}{|S|} \sum_{w \in S} T(w)$$

and S is the corresponding test set. The error rates of the algorithm HPWR are given together with 95% confidence intervals.

From the tables we can see that both algorithms HDWR and HPWR significantly outperform DWR on the sets of primitive elements with the error of HPWR being small (less than 1%). This shows that the fast checks are efficient reduction heuristics. The same picture holds for other sets. However, performance of HDWR deteriorates on sets S_1 and S_{10} , where occurrence of minimal elements is inevitable. We have already mentioned that in the case of a minimal element all of the Whitehead automorphisms have to be applied. The sizes of the sets of Whitehead elementary

automorphisms in free groups F_3 , F_4 and F_5 are $|\Omega_3| = 90$, $|\Omega_4| = 504$, $|\Omega_5| = 2550$ respectively. From the tables 7.2 and 7.3 we can see that the values of N_s in all cases is just a little greater than the size of Ω_n . This indicates that HDWR spends most of its automorphisms and, therefore, time on the elements of minimal length. On the contrary, algorithm HPWR seems to be able to avoid the exponential blowout by quickly recognizing minimal elements using the classifier WMIN. Note, that N_s computed for HPWR is smaller than $|\Omega_n|$ in all experiments.

To show that algorithm HPWR is applicable to groups of large ranks, we perform experiments with primitive elements in free groups F_{10} , F_{15} and F_{20} (see Table 7.4). We can see that HPWR was able to find solutions quickly with N_s growing very slow with the rank.

Conclusion 7.3.1. *The search heuristics described in Chapter 6 can be successfully applied for solving Whitehead Reduction problem. Probabilistic algorithm HPWR is very robust and can be used in groups with large ranks, where any other known algorithm fails to produce results due to the fact that the worst case is inevitable for most inputs. The computational advantage of HPWR increases when the rank of a free group increases. Indeed, HPWR performs about eleven times faster than DWR in F_3 and more than sixty times faster in F_5 .*

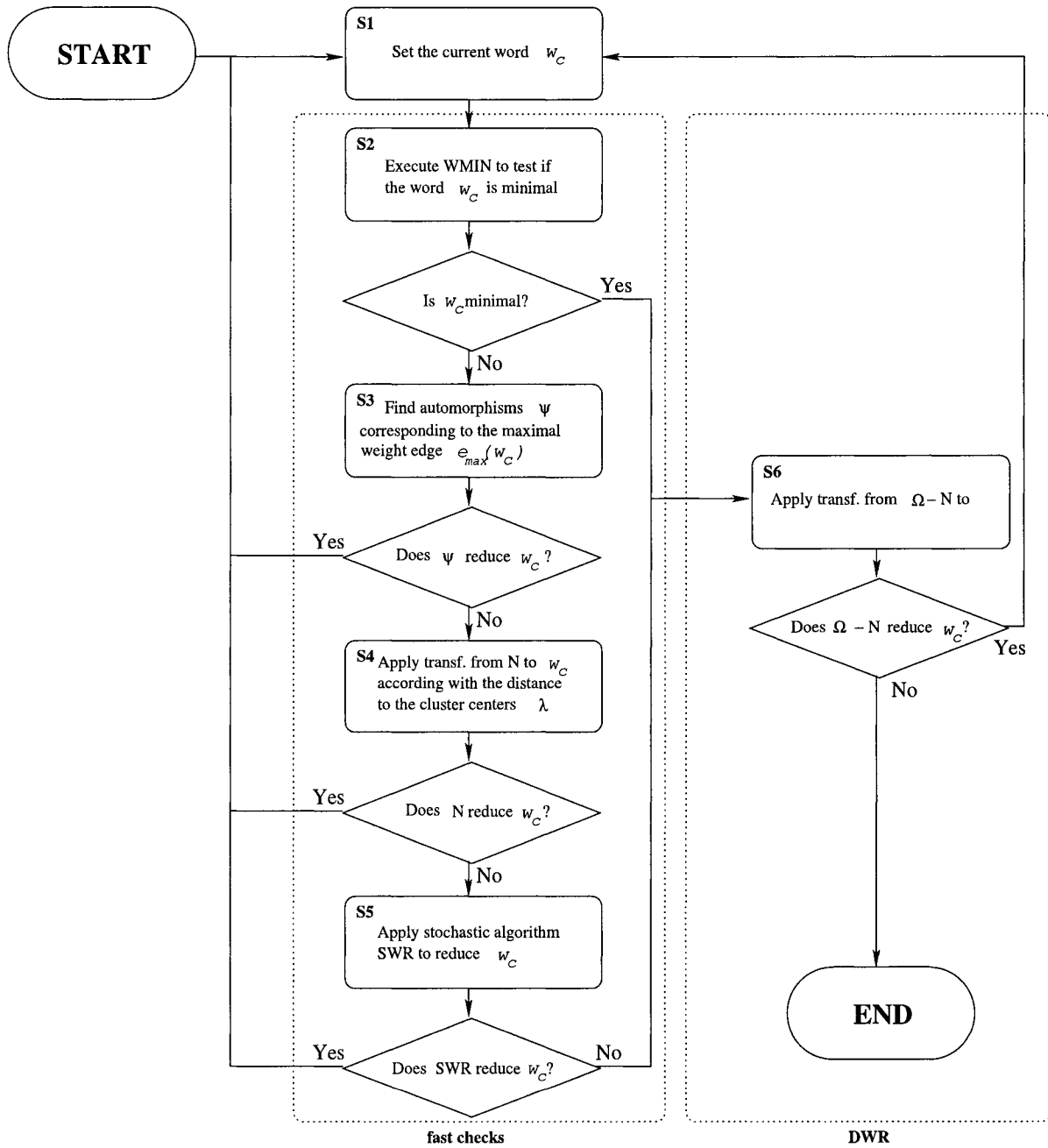


Figure 7.1: Flowchart diagram of the algorithm HDWR.

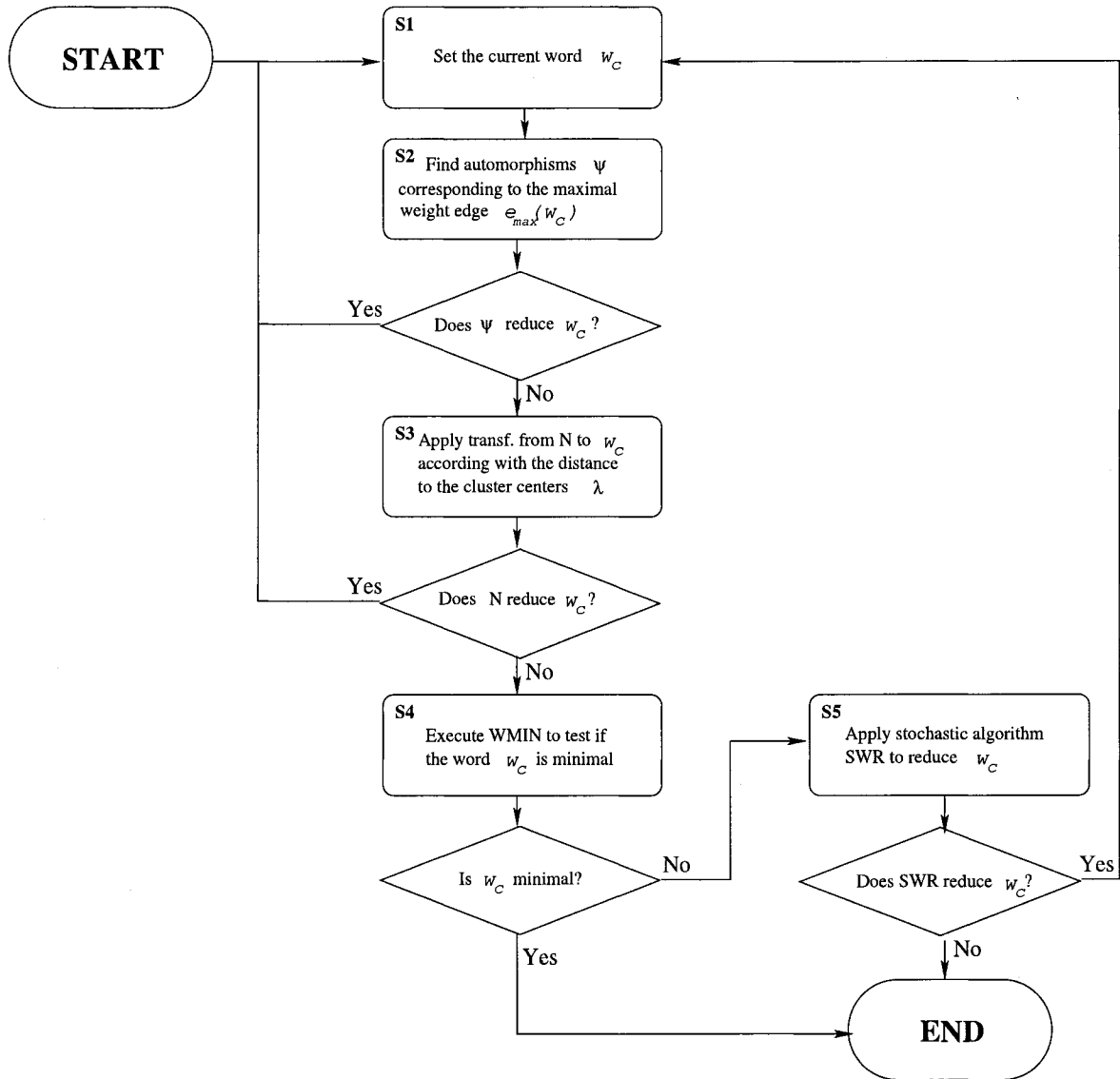


Figure 7.2: Flowchart diagram of the algorithm HPWR.

	N_s		N_{red}		$T_{avg, s}$		$E(\mathcal{A})$
$\mathcal{A}$	mean	std	mean	std	mean	std	
DWR	360.2	267.5	27.3	18.5	0.11	0.46	0.0
HDWR	41.2	29.1	24.2	15.0	0.01	0.05	0.0
HPWR	42.5	34.4	24.2	15.0	0.01	0.03	0.004 [0.0029,0.0065]

a) F_3 ;

	N_s		N_{red}		$T_{avg, s}$		$E(\mathcal{A})$
$\mathcal{A}$	mean	std	mean	std	mean	std	
DWR	2679.7	2356.5	57.5	37.3	2.03	8.75	0.0
HDWR	118.1	114.8	45.5	28.0	0.08	0.31	0.0
HPWR	127.2	184.0	45.5	28.0	0.06	0.2	0.005 [0.0038,0.0078]

b) F_4 ;

	N_s		N_{red}		$T_{avg, s}$		$E(\mathcal{A})$
$\mathcal{A}$	mean	std	mean	std	mean	std	
DWR	16319.9	20284.53	79.3	52.6	5.52	16.4	0.0
HDWR	276.5	539.4	58.9	38.6	0.12	0.29	0.0
HPWR	296.6	1157.7	58.0	35.6	0.13	0.26	0.003 [0.0016,0.0054]

c) F_5 ;

Table 7.1: Comparison of algorithms DWR, HDWR and HPWR on the test sets of primitive elements S_P in free groups F_3 , F_4 and F_5 , where N_s is the average number of elementary steps to find a minimal element, N_{red} the average number of length-reducing steps, T_{avg} is the average time (in seconds) spent on an input and $E(\mathcal{A})$ is the error rate of $\mathcal{A}$.

	N_s		N_{red}		$T_{avg, s}$		$E(\mathcal{A})$
$\mathcal{A}$	mean	std	mean	std	mean	std	
DWR	129.0	33.1	2.14	1.15	0.05	0.03	0.0
HDWR	117.5	10.4	1.69	0.81	0.04	0.03	0.0
HPWR	61.0	223.4	1.69	0.81	0.008	0.01	0.001 [0.00012,0.0018]

a) F_3 ;

	N_s		N_{red}		$T_{avg, s}$		$E(\mathcal{A})$
$\mathcal{A}$	mean	std	mean	std	mean	std	
DWR	734.9	188.3	3.24	1.72	0.30	0.20	0.0
HDWR	552.1	61.1	2.42	1.19	0.21	0.12	0.0
HPWR	153.3	647.7	2.43	1.29	0.02	0.03	0.001 [0.00012,0.0018]

b) F_4 ;

	N_s		N_{red}		$T_{avg, s}$		$E(\mathcal{A})$
$\mathcal{A}$	mean	std	mean	std	mean	std	
DWR	3541.3	908.2	4.28	2.19	1.45	1.05	0.0
HDWR	2601.6	341.7	3.29	1.73	0.70	0.41	0.0
HPWR	453.7	2168.6	3.28	1.70	0.05	0.09	0.002 [0.0013,0.0042]

c) F_5 ;

Table 7.2: Comparison of algorithms DWR, HDWR and HPWR on the test sets S_1 in free groups F_3 , F_4 and F_5 , where N_s is the average number of elementary steps to find a minimal element, N_{red} the average number of length-reducing steps, T_{avg} is the average time (in seconds) spent on an input and $E(\mathcal{A})$ is the error rate of $\mathcal{A}$.

	N_s		N_{red}		$T_{avg, s}$		$E(\mathcal{A})$
$\mathcal{A}$	mean	std	mean	std	mean	std	
DWR	203.5	86.5	8.45	5.49	0.12	0.13	0.00
HDWR	124.5	13.5	6.54	3.94	0.05	0.03	0.00
HPWR	78.8	257.0	6.54	3.94	0.01	0.03	0.0004 [0.0000,0.0014]

a) F_3 ;

	N_s		N_{red}		$T_{avg, s}$		$E(\mathcal{A})$
$\mathcal{A}$	mean	std	mean	std	mean	std	
DWR	1278.6	527.5	17.1	10.7	1.01	0.65	0.00
HDWR	569.7	67.5	11.6	7.16	0.36	0.33	0.00
HPWR	186.3	696.8	11.6	7.15	0.05	0.05	0.002 [0.0008,0.0034]

b) F_4 ;

	N_s		N_{red}		$T_{avg, s}$		$E(\mathcal{A})$
$\mathcal{A}$	mean	std	mean	std	mean	std	
DWR	7650.5	4468.0	27.1	17.8	5.87	8.81	0.00
HDWR	2650.5	342.9	17.1	10.8	1.06	0.63	0.00
HPWR	500.7	2195.3	16.9	10.6	0.10	0.16	0.002 [0.0012,0.0042]

c) F_5 ;

Table 7.3: Comparison of algorithms DWR, HDWR and HPWR on the test sets S_{10} in free groups F_3 , F_4 and F_5 , where N_s is the average number of elementary steps to find a minimal element, N_{red} the average number of length-reducing steps, T_{avg} is the average time (in seconds) spent on an input and $E(\mathcal{A})$ is the error rate of $\mathcal{A}$.

	N_s		N_{red}		$T_{avg, s}$		$E(\mathcal{A})$
$\mathcal{A}$	mean	std	mean	std	mean	std	
F_{10}	595.2	9195.5	55.5	37.0	0.20	0.58	0.001 [0.0000,0.0038]
F_{15}	671.1	883.1	106.1	55.5	1.03	0.59	0.000
F_{20}	736.3	874.4	128.4	61.8	2.80	1.41	0.000

Table 7.4: Performance of the algorithm HPWR on sets of primitive elements in free groups F_{10} , F_{15} and F_{20} , where N_s is the average number of elementary steps to find a minimal element, N_{red} the average number of length-reducing steps, T_{avg} is the average time (in seconds) spent on an input and $E(\mathcal{A})$ is the error rate of $\mathcal{A}$.

Conclusion

The purpose of this dissertation was to examine the Whitehead Minimization Problem from the statistical point of view by analyzing it with methods of statistical pattern recognition and exploratory data analysis. Our goal in doing this was to give an example of how such techniques, and corresponding methodologies, can be used in an analysis of a combinatorial problem in group theory.

We showed that starting from a naive approach, not using any particular results from group theory, one may proceed through a sequence of statistically significant discoveries and arrive at a better solution.

We also discussed specifics of experimental research in group theory, showing how to construct representative training datasets for the Whitehead Minimization Problem and suggested a number of test sets to insure unbiased experimental results. We suggested general space of feature representations of elements in free groups and introduced an algorithm to select economical and effective feature mappings.

We began by constructing a simple stochastic algorithm, SWR based on the genetic approach for solving the Whitehead Minimization problem. The experiments showed that SWR performs much better than the classic deterministic algorithm in free groups with a sufficiently large rank. Following our philosophy, we believe that certain unknown properties of the Whitehead Minimization problem must be

responsible for such effect.

Further observations allowed us to conclude that the upper bound on the number of iterations required for SWR to find a length reducing automorphism for a given non-minimal input does not depend on the inputs's length, but rather on the rank of a free group. Based on this conclusion we suggested replacing the length size function by Whitehead's Complexity function when measuring the time-complexity of the Length Reduction Problem.

We argued that the worst case in solving the Whitehead Minimization problem occurs when the word is minimal. To avoid exponential blowout in computations, one must provide an efficient solution for deciding whether or not an element is minimal. We solved this problem by developing a probabilistic classification system, WMIN. We showed that it is robust and performs well with a small classification error, even in groups with large ranks.

Further analysis of the classifiers suggested that elements are grouped in their feature space in compact regions and can be bounded by a hypersurface. Moreover, this groups might correspond to length-reducing automorphisms.

Using cluster analysis we showed that non-minimal elements could be classified in accordance with their length-reducing Nielsen automorphisms. Fast search heuristics that work on most input words were suggested. We showed that using appropriate heuristics in the algorithm DWR one can significantly reduce the complexity of the Whitehead Minimization Problem on most inputs with respect to group rank. Suggested heuristics reduce the average number of Whitehead automorphisms required to find a length-reducing automorphism for a given word.

The major contribution of this dissertation is the development of fast hybrid algorithms HDWR and HPWR based on the discovered search heuristics and probabilistic algorithms SWR and WMIN for solving the Whitehead Minimization problem. To reiterate, HPWR is very robust and can be used in groups with large ranks, where any other known algorithm fails to produce owing to the fact that the worst case is inevitable for most inputs.

Another contribution is the empirically discovered properties of non-minimal elements formulated as the Conjectures 6.2.1 and 6.3.1. These conjectures suggest that the generic set of non-minimal elements in a free group possesses a Nielsen length-reducing automorphism. Moreover, the feature vectors of weights of the Whitehead Graphs of elements from a free group are separated into "compact" regions in the corresponding space corresponding to a particular Nielsen automorphism, in the sense that all elements in the corresponding class can be reduced by that automorphism. We believe this is the first time meaningful rigorous conjectures in group theory were obtained using experimental simulations and statistical analysis of the problem.

Finally, we state some problems arising from the present research we would like to address in future writings, namely:

1. It certainly would be nice to prove Conjectures 6.2.1 and 6.3.1 which will allow us to have a complete example of working methodology of experimental mathematical discovery;
2. One of the follow-up works will concentrate on analyzing the "hard" examples of non-minimal words and try to obtain a deterministic solution to the Whitehead Minimization problem which is polynomial in the worst case;

3. We would like to see this or similar methodology applied to other problems in group theory and to hard combinatorial problems in general;
4. It would be useful to explore in greater detail the practical side of such an approach in cryptanalysis of non-commutative cryptographic schemes, for example.

Bibliography

- [1] Milton Abramowitz and Irene Stegun, *Handbook of mathematical functions with formulas, graphs, and mathematical table*, Dover Publications, Inc., 1972.
- [2] K. Appel and W. Haken, *Every planar map is four colorable. Part I. Discharging*, Illinois Journal of Mathematics **21** (1977), 429–490.
- [3] ———, *Every planar map is four colorable*, Contemporary Mathematics **98** (1989).
- [4] K. Appel, W. Haken, and J. Koch, *Every planar map is four colorable. Part II. Reducibility*, Illinois Journal of Mathematics **21** (1977), 491–567.
- [5] New York Group Theory Cooperative at CCNY, *MAGNUS: A graphically-oriented software system for computational group theory*, <http://zebra.sci.ccnycuny.edu/web/nygtc/software/aboutmagnus.html>, 1999.
- [6] David H. Bailey, Peter B. Borwein, and Simon Plouffe, *On the rapid computation of various polylogarithmic constants*, Mathematics of Computation (1997), 903–913.
- [7] C. E. Bonferroni, *Il calcolo delle assicurazioni su gruppi di teste*, Studi in Onore del Professore Salvatore Ortu Carboni (1935), 13–60.
- [8] Alexandre V. Borovik, Alexei G. Myasnikov, and Vladimir N. Remeslennikov, *Multiplicative measures on groups*, To appear, 2003.

- [9] Alexandre V. Borovik, Alexei G. Myasnikov, and Vladimir Shpilrain, *Measuring sets in infinite groups*, Computational and Statistical Group Theory, vol. 298, Amer. Math. Soc., Contemporary Math, 2002, pp. 21–42.
- [10] Jonathan M. Borwein and David H. Bailey, *Mathematics by Experiment: Plausible Reasoning in the 21st Century*, AK Peters Ltd, 2003.
- [11] Jonathan M. Borwein, David H. Bailey, and Roland Girgensohn, *Experimentation in Mathematics: Computational Paths to Discovery*, AK Peters Ltd, 2004.
- [12] G. E. P. Box, W. G. Hunter, and J. S. Hunter, *Statistics for experimenters*, Wiley, New York, 1978.
- [13] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and regression trees*, Wadsworth, Belmont, 1984.
- [14] B. Brown, J. Lovato, and K. Russell, *Dcdflib: A library of c routines for cumulative distribution functions, inverses, and other parameters*, Tech. report, Department of Biomathematics, The University of Texas, Houston, TX 77030, 1994, anonymous ftp: [odin.mda.uth.tmc.edu:/pub/unix/dcdflib.c.tar.Z](ftp://odin.mda.uth.tmc.edu/pub/unix/dcdflib.c.tar.Z).
- [15] Wray Buntine, *Learning classification trees*, Statistics and Computing **2** (1992), 63–73.
- [16] I.M. Chakravarti, R.G. Laha, and J. Roy, *Handbook of methods of applied statistics, volume i*, John Wiley and Sons, 1967.
- [17] B. A. Cipra, *Mathematics—Fermat’s Last Theorem Finally Yields*, Science **261** (1993), 32–33.
- [18] Ken De Jong, William Spears, and Diana Gordon, *Using Markov chains to analyze GAFOs*, Foundations of Genetic Algorithms **3** (1995), 115–138.

- [19] P. Diaconis and B. Efron, *Computer-intensive methods in statistics*, Scientific american **May** (1983), 116–130.
- [20] Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern classification*, 2nd ed., Wiley-Interscience, 2000.
- [21] B. Efron, *Estimating the error rate of a prediction rule: Improvement on cross-validation*, Journal of the American Statistical Association **78** (1983), 316–331.
- [22] B. Efron and R.J. Tibshirani, *An introduction to the bootstrap*, London: Chapman and Hall, 1993.
- [23] ———, *Improvements on cross-validation: The .632+ bootstrap method*, Journal of the American Statistical Association **92** (1997), 548–560.
- [24] W. Feller, *The fundamental limit theorems in probability*, Bull. Amer. Math. Soc. **51** (1945), 800–832.
- [25] K. Fukunaga, *Introduction to statistical pattern recognition*, Academic Press Inc., 1990.
- [26] F. Glover, *Future paths for integer programming and links to artificial intelligence*, Computers and operation research **13** (1986), 533–549.
- [27] ———, *The steepest ascent mildest descent heuristic for combinatorial programming*, Conf. on numerical methods in combinatorial optimization (1986).
- [28] ———, *Tabu search-part I*, ORSA Journal on Computing **1** (1989), no. 3, 190–260.
- [29] ———, *Tabu search-part II*, ORSA Journal on Computing **2** (1990), no. 1, 4–32.
- [30] Fred Glover and Manual Laguna, *Tabu search*, Kluwer Academic Publishers, 1997.

- [31] David E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley Publishing Company, 1989.
- [32] P. Good, *Permutation tests: A practical guide to resampling methods for testing hypotheses*, Springer-Verlag, New York, 2000.
- [33] Brian (Editor) Gough, *GNU scientific library reference manual - second edition*, Network Theory Ltd., 2003.
- [34] C. Goutte, *Note on free lunches and cross-validation*, Neural Computation **9** (1997), 1211–1215.
- [35] T. C. Hales and S. P. Ferguson, Ann. Math. (2003).
- [36] S. Haykin, *Neural networks: A comprehensive foundation*, Prentice Hall, Upper Saddle River, NJ, 1999.
- [37] J.H. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press (Ann Arbor), 1975.
- [38] I. Kapovich, P. Schupp, and V. Shpilrain, *Generic properties of whitehead's algorithm, stabilizers in $\text{aut}(f_k)$ and one-relator groups*, Preprint, 2003.
- [39] Bilal Khan, *Automorphic orbits in f_2* , Ph.D. Thesis, GC CUNY, 2003.
- [40] O. Kharlampovich and A. Myasnikov, *Tarski's problem about the elementary theory of free groups has a positive solution*, Electron. Res. Announc. Amer. Math. Soc. **4** (1998), 101–108.
- [41] Scott Kirkpatrick, C. Daniel Gelatt Jr., and Mario P. Vecchi, *Optimization by simulated annealing*, Science **220** (1983), no. 4598, 671–680.
- [42] Donghi Lee, *The complexity of whitehead's algorithm*, In preparation.

- [43] L. Levin, *Average case complete problems*, SIAM Journal on Computing **15** (1986), 285–286.
- [44] R. Lyndon and P. Schupp, *Combinatorial group theory*, Series of Modern Studies in Math. 89. Springer-Verlag, 1977.
- [45] Wendy L. Martinez and Angel R. Martinez, *Computational statistics handbook with MATLAB*, CRC Press, 2001.
- [46] Geoffrey J. McLachlan, *Discriminant analysis and statistical pattern recognition*, Wiley-Interscience, 1992.
- [47] Tzuong Tsieng Moh, *A public key system with signature and master key functions*, Communications in Algebra **27** (1999), no. 5, 2207–2222.
- [48] Alexei G. Myasnikov and Vladimir Shpilrain, *Automorphic orbits in free groups*, Journal of Algebra **269** (2002), 18–27.
- [49] J Nielsen, *Die isomorphismengruppe der freien gruppen*, Math. Ann. **91** (1924), 169–209.
- [50] A.E. Nix and M.D. Vose, *Modelling genetic algorithms with Markov chains*, Annals of Mathematics and Artificial Intelligence **5** (1992), 79 – 88.
- [51] Karl Pearson, *On a criterion that a given system of deviation from the probable in the case of correlated system of variables is sauch that it can reasonably be supposed to have arisen from random sampling*, Philosophical Magazine **50** (1900), 157–175.
- [52] F. Pesarin, *On a nonparametric combination method for dependent permutation tests with applications*, Psychotherapy Psychosomatics **54** (1935), 172–179.
- [53] D.T. Pham and D. Karaboga, *Intelligent optimization techiques*, Springer-Verlag, 2000.

- [54] M. Plutowski, S. Sakata, and H. White, *Cross-validation estimates imse*, Advances in Neural Information Processing Systems **6** (1994), 391–398.
- [55] J.R. Quinlan, *C4.5: Programs for machine learning*, Morgan Kaufmann, San Francisco, 1993.
- [56] N. Radcliffe and P. Surr, *Fundamental limitations on search algorithms: Evolutionary computing in perspective*, Lecture notes of computer science, Springer-Verlag, 1995.
- [57] C. R. Reeves, *Genetic algorithms for the operations research*, INFORMS Journal on Computing **9** (1997), no. 3, 231–250.
- [58] Colin R. Reeves (ed.), *Modern heuristic techniques for combinatorial problems*, Halsted Press, 1993.
- [59] H. Scheffe, *The analysis of variance*, John Wiley and Sons, New York, 1959.
- [60] M. A. Stephens, *EDF statistics for goodness of fit and some comparisons*, Journal of the American Statistical Association **69** (1974), 730–737.
- [61] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, Academic Press, 1999.
- [62] H.L. Van Trees, *Detection, estimation, and modulation theory, part i: Detection, estimation, and linear modulation theory*, John Wiley and Sons, 1968.
- [63] V.N. Vapnik, *Statistical learning theory*, John Wiley and Sons, New York, 1998.
- [64] ———, *The nature of statistical learning theory*, Springer-Verlag, New York, 2000.
- [65] M.D. Vose, *Random heuristic search*, Theoretical Computer Science (1999), 103–142.

- [66] Jie Wang, *Average-case computational complexity theory*, Complexity Theory Retrospective II (L. Hemaspaandra and A. Selman, eds.), Springer, 1997, pp. 295–328.
- [67] S.M. Weiss and C.A. Kulikowski, *Computer systems that learn*, Morgan Kaufmann, 1991.
- [68] J. H. C. Whitehead, *On equivalent sets of elements in a free group*, Annals of Mathematics **37** (1936), no. 4, 782–800.
- [69] David H. Wolpert and William G. Macready, *No free lunch theorems for search*, Tech. Report SFI-TR-95-02-010, 1995.
- [70] D.H. Wolpert and W.G. Macready, *No free lunch theorems for optimization*, IEEE Transactions on Evolutionary Computation **1** (1997), 67–82.