

**AN EMPIRICAL STUDY OF MODULE ORDER
MODELS**

by

Boonlit Adipat

A Thesis Submitted to the Faculty of

The College of Engineering

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

Florida Atlantic University

Boca Raton, Florida

August 2001

AN EMPIRICAL STUDY OF MODULE ORDER MODELS

by

Boonlit Adipat

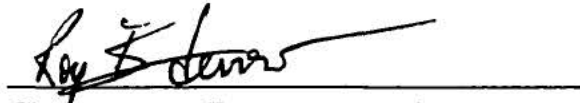
This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Taghi M. Khoshgoftaar, Department of Computer Science and Engineering, and has been approved by the members of his supervisory committee. It was submitted to the faculty of The College of Engineering and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

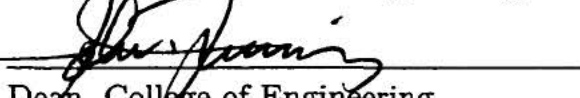
SUPERVISORY COMMITTEE:

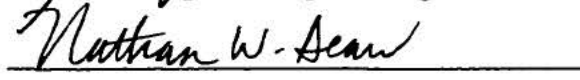

Thesis Advisor

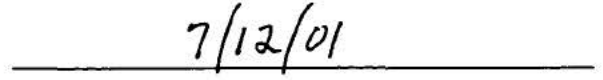





Chairperson, Department of
Computer Science and Engineering


Dean, College of Engineering


Vice Provost


Date

ACKNOWLEDGMENTS

I would like to express my gratitude and thanks to my advisor, Dr. Taghi M. Khoshgoftaar, for his guidance, invaluable comments and great encouragement offered during my thesis work. His profound knowledge in the field of software reliability and quality engineering were immensely help me to complete my work.

I thank Dr. Eduardo Fernandez and Dr. Marty Solomon for willing to serve on my thesis committee and reviewing my thesis. I would like to give the great thank to Erik Geleyn for patiently reviewing my thesis, Kehan Gao for advising me how to use \LaTeX I also thank Naeem Seliya and Nandini Sundaresh for their research work that made my work possible.

I thank to my previous advisor, Suwat Pattaramalai and all of my friends in ESEL lab, Sajan, Dhaval, Bhavana, Yongbin and in Thai Student Organization, Sangthen Chirdchid, Narongrit Butrieng, Settapong and Usanee Malisuwan

I would like to share the great pride and express the best gratitude to my parents, Yuttana and Nuchakorn Adipat for their moral support, precious advice, and sacrifices they have made throughout my studies.

I appreciate the moral support from my very special friend Alisa Kasintorn.

ABSTRACT

Author: Boonlit Adipat
Title: An Empirical Study of Module Order Models
Institution: Florida Atlantic University
Thesis Advisor: Dr. Taghi M. Khoshgoftaar
Degree: Master of Science
Year: 2001

Most software reliability approaches classify modules as fault-prone or not fault-prone by way of a predetermined threshold. However, it may not be practical to predefine a threshold because the amount of resources for reliability enhancement may be unknown. Therefore, a module-order model (MOM) predicting the rank-order of modules can be used to solve this problem. The objective of this research is to make an empirical study of MOMs based on five different underlying quantitative software quality models. We examine the benefits of principal components analysis with MOM and demonstrate that better accuracy of underlying techniques does not always yield better performance with MOM. Three case studies of large industrial software systems were conducted. The results confirm that MOM can create efficient models using different underlying techniques that provide various accuracy when predicting a quantitative software quality factor over the data sets.

To my parents

CONTENTS

TABLES	vii
FIGURES	ix
1 INTRODUCTION	1
2 SOFTWARE METRICS	7
2.1 Introduction	7
2.2 Software Metrics Used in This Study	9
2.3 LLTS metrics	10
2.3.1 LLTS product Metrics	10
2.3.2 LLTS Process Metrics	14
2.3.3 LLTS Execution Metrics	15
2.4 NT metrics	15
2.5 LNTS metrics	16
3 METHODOLOGY	19
3.1 Module-Order Modeling	19
3.2 Classification	26
3.3 SMART	27
3.4 Principal Components Analysis	31
3.5 Model Performance Evaluation	33
3.6 Underlying quantitative models	33
3.6.1 Case-Based Reasoning	33
3.6.2 Multiple Linear Regression	36
3.6.3 Artificial Neural Network	37
3.6.4 Tree model	39

3.6.5	CART	40
3.6.6	SPLUS	41
4	EXPERIMENTS	43
4.1	Case Study Methodology	43
4.2	System Description	45
4.2.1	LLTS System Description	45
4.2.2	NT System Description	48
4.2.3	LNTS System Description	49
4.3	Experiments on LLTS	51
4.3.1	Experiments on LLTS-RAW	55
4.3.2	Experiment on LLTS-PCA	76
4.4	Experiment on NT	105
4.4.1	Experiment on NT-RAW	106
4.4.2	Experiment on NT-PCA	116
4.5	Experiment on LNTS	125
4.5.1	Experiment on LNTS-RAW	126
4.5.2	Experiment on LNTS-PCA	135
4.6	Comparing module-order models based on RAW and PCA metrics .	146
4.6.1	Comparing module-order models for LLTS	147
4.6.2	Comparing module-order models of NT	151
4.6.3	Comparing module-order models for LNTS	154
5	CONCLUSIONS	160
5.1	Overview	160
5.2	Future Work	163
	BIBLIOGRAPHY	164

TABLES

2.1	Software Product Metrics of LLTS data	11
2.2	Software Process Metrics of LLTS data	12
2.3	Software Execution Metrics of LLTS data	12
2.4	NT System Profile	16
2.5	NT Product Metrics	17
2.6	LNTS System Profile	18
2.7	LNTS Product Metrics	18
3.1	Effectiveness and efficiency	28
4.1	Factor Pattern for Principal Components of Product Metrics for LLTS data set	47
4.2	Factor Pattern for Principal Components of Design Product Metrics for NT data set	49
4.3	Factor Pattern for Principal Components of Software Metrics for LNTS data set	50
4.4	Example of module-order modeling result	52
4.5	Presentation outline for LLTS data	55
4.6	LLTS-RAW, Comparative accuracy of underlying quantitative models	56

4.7	LLTS-PCA, Comparative accuracy of underlying quantitative models	81
4.8	Presentation outline for NT data	105
4.9	NT-RAW, Comparative accuracy of underlying quantitative models	106
4.10	NT-PCA, Comparative accuracy of underlying quantitative models	116
4.11	Presentation outline for LNTS data	126
4.12	LNTS-RAW, Comparative accuracy of underlying quantitative models	127
4.13	LNTS-PCA, Comparative accuracy of underlying quantitative models	137

FIGURES

3.1	Module-order model operation	21
3.2	Smart Architecture	29
3.3	MOM page in SMART	30
3.4	Example of Tree model in purpose of classification	39
4.1	Example of Alberg diagram	53
4.2	Example of Performance diagram	53
4.3	Alberg diagram for LLTS-RAW release 2: CBR, MLR, ANN	60
4.4	Close view of Alberg diagram for LLTS-RAW release 2: CBR, MLR, ANN	60
4.5	Performance of LLTS-RAW release 2: CBR, MLR, ANN	61
4.6	Alberg diagram for LLTS-RAW release 3: CBR, MLR, ANN	61
4.7	Close view of Alberg diagram for LLTS-RAW release 3: CBR, MLR, ANN	62
4.8	Performance of LLTS-RAW release 3: CBR, MLR, ANN	62
4.9	Alberg diagram for LLTS-RAW release 4: CBR, MLR, ANN	63
4.10	Close view of Alberg diagram for LLTS-RAW release 4: CBR, MLR, ANN	63
4.11	Performance of LLTS-RAW release 4: CBR, MLR, ANN	64

4.12	Alberg diagram for LLTS-RAW release 2: CART-LS, CART-LAD, SPLUS	66
4.13	Close view of Alberg diagram for LLTS-RAW release 2: CART-LS, CART-LAD, SPLUS	66
4.14	Performance of LLTS-RAW release 2: CART-LS, CART-LAD, SPLUS	67
4.15	Alberg diagram for LLTS-RAW release 3: CART-LS, CART-LAD, SPLUS	67
4.16	Close view of Alberg diagram for LLTS-RAW release 3: CART-LS, CART-LAD, SPLUS	68
4.17	Performance of LLTS-RAW release 3: CART-LS, CART-LAD, SPLUS	68
4.18	Alberg diagram for LLTS-RAW release 4: CART-LS, CART-LAD, SPLUS	69
4.19	Close view of Alberg diagram for LLTS-RAW release 4: CART-LS, CART-LAD, SPLUS	69
4.20	Performance of LLTS-RAW release 4: CART-LS, CART-LAD, SPLUS	70
4.21	Alberg diagram of LLTS-RAW release 2: CART-LS, CART-LAD, SPLUS and CBR	71
4.22	Close view of Alberg diagram of LLTS-RAW release 2: CART-LS, CART-LAD, SPLUS and CBR	71
4.23	Performance of LLTS-RAW release 2: CART-LS, CART-LAD, SPLUS and CBR	72
4.24	Alberg diagram for LLTS-RAW release 3: CART-LS, CART-LAD, SPLUS and CBR	72
4.25	Close view of Alberg diagram for LLTS-RAW release 3: CART-LS, CART-LAD, SPLUS and CBR	73

4.26	Performance of LLTS-RAW release 3: CART-LS, CART-LAD, SPLUS and CBR	73
4.27	Alberg diagram for LLTS-RAW release 4: CART-LS, CART-LAD, SPLUS and CBR	74
4.28	Close view of Alberg diagram for LLTS-RAW release 4: CART-LS, CART-LAD, SPLUS and CBR	74
4.29	Performance of LLTS-RAW release 4: CART-LS, CART-LAD, SPLUS and CBR	75
4.30	Alberg diagram of LLTS-RAW release 2: CART-LS, CART-LAD	77
4.31	Close view of Alberg diagram for LLTS-RAW release 2: SPLUS, CART-LAD	77
4.32	Performance of LLTS-RAW release 2: CART-LS, CART-LAD	78
4.33	Alberg diagram for LLTS-RAW release 3: SPLUS, CART-LAD	78
4.34	Close view of Alberg diagram for LLTS-RAW release 3: SPLUS, CART-LAD	79
4.35	Performance of LLTS-RAW release 3: SPLUS, CART-LAD	79
4.36	Alberg diagram for LLTS-RAW release 4: CART-LS, CART-LAD	80
4.37	Close view of Alberg diagram for LLTS-RAW release 4: SPLUS, CART-LAD	80
4.38	Performance of LLTS-RAW release 4: CART-LS, CART-LAD	81
4.39	Alberg diagram for LLTS-PCA release 2: CBR, MLR, ANN	84
4.40	Close view of Alberg diagram for LLTS-PCA release 2: CBR, MLR, ANN	85
4.41	Performance of LLTS-PCA release 2: CBR, MLR, ANN	85
4.42	Alberg diagram for LLTS-PCA release 3: CBR, MLR, ANN	86

4.43	Close view of Alberg diagram for LLTS-PCA release 3: CBR, MLR, ANN	86
4.44	Performance of LLTS-PCA release 3: CBR, MLR, ANN	87
4.45	Alberg diagram for LLTS-RAW release 4: CBR, MLR, ANN	87
4.46	Close view of Alberg diagram for LLTS-PCA release 4: CBR, MLR, ANN	88
4.47	Performance of LLTS-PCA release 4: CBR, MLR, ANN	88
4.48	Alberg diagram for LLTS-PCA release 2: CART-LS, CART-LAD, SPLUS	90
4.49	Close view of Alberg diagram for LLTS-PCA release 2: CART-LS, CART-LAD, SPLUS	90
4.50	Performance of LLTS-PCA release 2: CART-LS, CART-LAD, SPLUS	91
4.51	Alberg diagram for LLTS-PCA release 3: CART-LS, CART-LAD, SPLUS	91
4.52	Close view of Alberg diagram for LLTS-PCA release 3: CART-LS, CART-LAD, SPLUS	92
4.53	Performance of LLTS-PCA release 3: CART-LS, CART-LAD, SPLUS	92
4.54	Alberg diagram for LLTS-PCA release 4: CART-LS, CART-LAD, SPLUS	93
4.55	Close view of Alberg diagram for LLTS-PCA release 4: CART-LS, CART-LAD, SPLUS	93
4.56	Performance of LLTS-PCA release 4: CART-LS, CART-LAD, SPLUS	94
4.57	Alberg diagram for LLTS-PCA release 2: CART-LS, CART-LAD, SPLUS and ANN	95

4.58	Close view of Alberg diagram for LLTS-PCA release 2: CART-LS, CART-LAD, SPLUS and ANN	96
4.59	Performance of LLTS-PCA release 2: CART-LS, CART-LAD, SPLUS and ANN	96
4.60	Alberg diagram for LLTS-PCA release 3: CART-LS, CART-LAD, SPLUS and ANN	97
4.61	Close view of Alberg diagram for LLTS-PCA release 3: CART-LS, CART-LAD, SPLUS and ANN	97
4.62	Performance of LLTS-PCA release 3: CART-LS, CART-LAD, SPLUS and ANN	98
4.63	Alberg diagram for LLTS-PCA release 4: CART-LS, CART-LAD, SPLUS and ANN	98
4.64	Close view of Alberg diagram for LLTS-PCA release 4: CART-LS, CART-LAD, SPLUS and ANN	99
4.65	Performance of LLTS-PCA release 4: CART-LS, CART-LAD, SPLUS and ANN	99
4.66	Alberg diagram for LLTS-PCA release 2: CART-LS, CART-LAD	101
4.67	Close view of Alberg diagram for LLTS-PCA release 2: CART-LS, CART-LAD	101
4.68	Performance of LLTS-PCA release 2: CART-LS, CART-LAD . . .	102
4.69	Alberg diagram for LLTS-PCA release 3: MLR, CART-LAD . . .	102
4.70	Close view of Alberg diagram for LLTS-PCA release 3: CART-LS, CART-LAD	103
4.71	Performance of LLTS-PCA release 3: MLR, CART-LAD	103
4.72	Alberg diagram for LLTS-PCA release 4: SPLUS, CBR	104

4.73	Close view of Alberg diagram for LLTS-PCA release 4: SPLUS, CBR	104
4.74	Performance of LLTS-PCA release 4: SPLUS, CBR	105
4.75	Alberg diagram for NT-RAW: CBR, MLR, ANN	108
4.76	Close view of Alberg diagram for NT-RAW: CBR, MLR, ANN . .	109
4.77	Performance of NT-RAW: CBR, MLR, ANN	109
4.78	Alberg diagram for NT-RAW: CART-LS, CART-LAD, SPLUS . .	111
4.79	Close view of Alberg diagram for NT-RAW: CART-LS, CART-LAD, SPLUS	111
4.80	Performance of NT-RAW: CART-LS, CART-LAD, SPLUS	112
4.81	Alberg diagram for NT-RAW: CART-LS, CART-LAD, SPLUS and CBR	113
4.82	Close view of Alberg diagram for NT-RAW: CART-LS, CART-LAD, SPLUS and CBR	113
4.83	Performance of NT-RAW: CART-LS, CART-LAD, SPLUS and CBR	114
4.84	Alberg diagram for NT-RAW: CBR, MLR	114
4.85	Close view of Alberg diagram for NT-RAW: CBR, MLR	115
4.86	Performance of NT-RAW: CBR, MLR	115
4.87	Alberg diagram for NT-PCA: CBR, MLR, ANN	118
4.88	Close view of Alberg diagram for NT-PCA: CBR, MLR, ANN . .	119
4.89	Performance of NT-PCA: CBR, MLR, ANN	119
4.90	Alberg diagram for NT-PCA: CART-LS, CART-LAD, SPLUS . .	121

4.91	Close view of Alberg diagram for NT-PCA: CART-LS, CART-LAD, SPLUS	121
4.92	Performance of NT-PCA: CART-LS, CART-LAD, SPLUS	122
4.93	Alberg diagram for NT-PCA: CART-LS, CART-LAD, SPLUS and MLR	123
4.94	Close view of Alberg diagram for NT-PCA: CART-LS, CART-LAD, SPLUS and MLR	124
4.95	Performance of NT-PCA: CART-LS, CART-LAD, SPLUS and MLR	124
4.96	Alberg diagram of NT-PCA: MLR, CART-LAD	125
4.97	Close view of Alberg diagram for NT-PCA: MLR, CART-LAD	125
4.98	Performance of NT-PCA: MLR, CART-LAD	126
4.99	Alberg diagram for LNTS-RAW: CBR, MLR, ANN	129
4.100	Close view of Alberg diagram for LNTS-RAW: CBR, MLR, ANN	129
4.101	Performance of LNTS-RAW: CBR, MLR, ANN	130
4.102	Alberg diagram for LNTS-RAW: CART-LS, CART-LAD, SPLUS	131
4.103	Close view of Alberg diagram for LNTS-RAW: CART-LS, CART-LAD, SPLUS	132
4.104	Performance of LNTS-RAW: CART-LS, CART-LAD, SPLUS	132
4.105	Alberg diagram for LNTS-RAW: CART-LS, CART-LAD, SPLUS and CBR	133
4.106	Close view of Alberg diagram for LNTS-RAW: CART-LS, CART-LAD, SPLUS and CBR	134
4.107	Performance of LNTS-RAW: CART-LS, CART-LAD, SPLUS and CBR	134

4.108	Alberg diagram for LNTS-RAW: CART-LS, CART-LAD	136
4.109	Close view of Alberg diagram for LNTS-RAW: CART-LS, CART-LAD	136
4.110	Performance of LNTS-RAW: CART-LS, CART-LAD	137
4.111	Alberg diagram for LNTS-PCA: CBR, MLR, ANN	139
4.112	Close view of Alberg diagram for LNTS-PCA: CBR, MLR, ANN .	139
4.113	Performance of LNTS-PCA: CBR, MLR, ANN	140
4.114	Alberg diagram for LNTS-PCA: CART-LS, CART-LAD, SPLUS	141
4.115	Close view of Alberg diagram for LNTS-PCA: CART-LS, CART-LAD, SPLUS	142
4.116	Performance of LNTS-PCA: CART-LS, CART-LAD, SPLUS . .	142
4.117	Alberg diagram for LNTS-PCA: CART-LS, CART-LAD, SPLUS and CBR	143
4.118	Close view of Alberg diagram for LNTS-PCA: CART-LS, CART-LAD, SPLUS and CBR	144
4.119	Performance of LNTS-PCA: CART-LS, CART-LAD, SPLUS and CBR	144
4.120	Alberg diagram for LNTS-PCA: CART-LS, CART-LAD	145
4.121	Close view of Alberg diagram for LNTS-PCA: CART-LS, CART-LAD	145
4.122	Performance of LNTS-PCA: CART-LS, CART-LAD	146
4.123	Alberg diagram for LLTS PCA and RAW comparison release 4: CBR	147
4.124	Alberg diagram for LLTS PCA and RAW comparison release 2: MLR	148

4.125	Alberg diagram for LLTS PCA and RAW comparison release 2: ANN	148
4.126	Alberg diagram for LLTS PCA and RAW comparison release 2: CART-LS	149
4.127	Alberg diagram for LLTS PCA and RAW comparison release 2: CART-LAD	150
4.128	Alberg diagram for LLTS PCA and RAW comparison release 3: CART-LAD	150
4.129	Alberg diagram for LLTS PCA and RAW comparison release 4: SPLUS	151
4.130	Alberg diagram for NT PCA and RAW comparison: CBR	152
4.131	Alberg diagram for NT PCA and RAW comparison: MLR	153
4.132	Alberg diagram for NT PCA and RAW comparison: ANN	153
4.133	Alberg diagram for NT PCA and RAW comparison: CART-LS . .	154
4.134	Alberg diagram for NT PCA and RAW comparison: CART-LAD	155
4.135	Alberg diagram for NT PCA and RAW comparison: SPLUS . . .	155
4.136	Alberg diagram for LNTS PCA and RAW comparison: CBR . . .	156
4.137	Alberg diagram for LNTS PCA and RAW comparison: MLR . . .	156
4.138	Alberg diagram for LNTS PCA and RAW comparison: ANN . . .	157
4.139	Alberg diagram for LNTS PCA and RAW comparison: CART-LS	157
4.140	Alberg diagram for LNTS PCA and RAW comparison: CART-LAD	158
4.141	Alberg diagram for LNTS PCA and RAW comparison: SPLUS . .	158

Chapter 1

INTRODUCTION

The importance of software engineering has been increasing in industrials, businesses and organization all over the world. High assurance of software quality is required because of the large amount of monetary loss or even unassessed cost to human lives due to a potential software failure. Consequently, reliability becomes an undeniable ingredient while developing software products. However, a reliability process involves time consumption, budget and quality standards. The limited time and high cost reduces the quality of software testing, which is the essential process in the software development cycle. To solve the problem, we may focus on the modules that are most likely to be faulty and apply reliability-enhancement activities to them [6]. A software fault is a defect in an operational product causing the software failure [5].

While performing the reliability enhancement process, a software quality model is created to help to predict the number of faults in modules early in the life cycle. Numerous researches have focused on classification models to identify fault-prone and not fault-prone modules early in the life cycle [3, 14, 16, 22]. To

define the fault-prone or not fault-prone modules, we have to determine a threshold before modeling. However, it's hard to define the threshold at the time of modeling because of the unspecified amount of resources for the reliability-improvement effort. Therefore, a module-order model (MOM) predicting the rank-order of modules has been proposed to alleviate the problem of classifying fault-prone or not fault-prone modules [10].

A module-order model (MOM) predicts the rank-order of modules based on a software quality factor such as number of faults, and uses the selected cutoff rank for reliability enhancement. The modules above the cutoff point are classified as fault-prone modules, otherwise the modules are not fault-prone. A module-order model consists of an underlying quantitative model that produces a prediction of the quality factor and an ordering of the modules by using the predicted quality factor. The product and process metrics are used as inputs to the underlying quantitative models. Then, a module-order model retrieves the predicted variable from the underlying techniques as input.

Preliminary research [10] gave the definition of the module-order model and a method to build and evaluate the model. Multiple Linear Regression was the underlying quantitative model. Another study [12] observed the comparative result of module-order model with nonparametric discriminant analysis for the purpose of classification . The underlying quantitative model applied in this study was also Multiple Linear Regression. The study used module-order modeling to build the

rank-order of modules based on the number of faults. Then, the defined threshold was provided to compute the misclassification rates. This result was compared to other misclassification rates obtained by applying nonparametric discriminant analysis.

Both researches used the same two empirical case studies to evaluate the performance of module-order models. The conclusion from these two researches demonstrated that module-order modeling is very useful to use when the threshold can not be appropriately defined at the time of modeling. The results were consistently effective and robust in the two different projects, and MOM even performs better than nonparametric discriminant analysis in view of classification. In addition, a module-order model presented good performances even though the underlying quantitative model produced the predicted dependent quality factor with poor accuracy.

As previously described, the prior studies were only based on one underlying technique. However, any underlying quantitative model can be applied to module-order modeling as long as it uses at least an ordinal scale [18]. Therefore, our study will complete prior research about module-order modeling based on a variety of underlying quantitative models. The main objectives of this thesis can be summarized as the following items.

- Study the impact of different underlying quantitative models on module-order modeling. In this study, we focus on the following five different underlying

techniques: Case-Based Reasoning (CBR), Multiple Linear Regression (MLR), Artificial Neural Network (ANN), CART-Least Square (CART-LS), CART-Least Absolute Deviation (CART-LAD) and SPLUS.

- Study whether principal components analysis can give the benefits when module-order modeling.
- Verify the hypothesis that better accuracy of underlying techniques does not ensure better performance in module-order modeling.

The first scope of our research is to perform a case study of module-order modeling using different underlying techniques. A module-order model can use any method to retrieve the predicted quality factor. We used graphical presentation to observe the behaviors and performances of the different module-order models. It was found that the module-order models based on CBR, MLR and ANN had very similar behaviors and performances. The diagrams show the very close trend of lines along the considered ranges. In contrast, the tree modeling techniques, CART-LS, CART-LAD and SPLUS, present different results. The graphs illustrated the separately trend of lines, especially for the two CART methods. They present the varying behavior and performance along the considered range. Further, when comparing tree-modeling with non tree-modeling methods, we found that SPLUS performs close to CBR, MLR and ANN when module-order modeling.

The second scope investigates the benefits of using principal components analysis [28] when module-order modeling. We compared the module-order models built using PCA with ones using RAW metrics. It was observed that the module-order models using PCA had very close behaviors to the ones using RAW metrics for CBR, MLR and ANN. For tree-modeling techniques, the models built using RAW are usually better than the ones using PCA metrics. Therefore, it was concluded that the use of principal components analysis did not improve the module-order models.

The third scope of our research is to verify that better prediction accuracy does not guarantee better performance when module-order modeling. In prior researches [12, 10], two different data sets were used to build the model. The results of the underlying quantitative prediction used for module-order modeling were different. The accuracy of the predicted quality factor from one data set was much better than one from the other. However, the performance of module-order models were about the same. In this study, the variation of fundamental quantitative models and incremental case studies were added. The accuracy of each underlying quantitative model in each data set was compared before applying module-order modeling. After conducting the experiments, we found several evidences to confirm our hypothesis. The techniques with the best prediction didn't yield better performances than the other underlying models when module-order modeling.

This study used three large data sets as materials for the case studies. The

first case is a very large legacy telecommunications system written in a high level language similar to PASCAL. The second one is a large network telecommunications system, and the last one is also a network telecommunications system. All of these three software systems have the complete characteristics making them suitable for a case study based on the software engineering community standard [30]. The experiment was conducted on both raw and principal component metrics for all three data sets. The tool used in this thesis is the Software Measurement Analysis Reliability Toolkit, SMART, developed at the Empirical Software Engineering Lab (ESEL), Florida Atlantic University [13].

Looking through the layout of this study. This thesis consists of five chapters. Chapter 1, introduction, describes the objectives and history of previous researches in the area of interest. Chapter 2, Software metrics, gives the details of different metrics used in this study. Chapter 3, methodology, discusses the various methodologies involved in this study. Module-order modeling is explained in deep detail. Other underlying modeling techniques and some algorithms are also briefly described. Chapter 4, experiment, presents the case study experiments and their results. Chapter 5, conclusion, analyzes the result and concludes with the lesson learned in this study.

Chapter 2

SOFTWARE METRICS

This Chapter explains the definition and the importance of software metrics, including the information about the metrics used in the three systems in this study. Those three systems are the large legacy telecommunications system (LLTS), the network telecommunications (NT), and the large network telecommunications system (LNTS).

2.1 Introduction

Software metrics are the results of software measurement activities used to evaluate the performance, reliability and quality of the software processes and products. The attributes of software metrics are distinguished into 2 groups, internal and external attributes [5].

- Internal attributes are defined as attributes that can be measured by observing the characteristic of the process and product separated from the system's environment such as size (line of codes), complexity (number of decision points)

- External attributes are measured to determine how the process and product interacts with its environment. These attributes can be measured only when the code is executed, for example, the number of failure experienced by users.

For software metrics, there is no definite conclusion on how to group the metrics. In this study, software metrics can be categorized into 3 classes: process, product and execution [5].

Process metrics are the collections of software development activities associated with time. The activities in the particular process are arranged according to a time schedule. This means one activity cannot begin unless the previous activity is finished. Therefore, process metrics measure the attributes during the development process. Examples of this kind of metric can be any activity in the development period such as history of faults discovered and corrections, time used to fix errors or even the records of programmers, etc.

Product metrics are the artifact or document observed at the present time on the software product. The program's source code is classified in this group. Product metrics are not concerned with the development process. In contrast, they focus on the structure of the software. Example of product metrics are program size, number of calls to other modules, number of loops, etc.

Execution metrics are the gathered information when the software is executing in the real operating environment. Examples of execution metrics are the duration of operation and deployment of the product.

Standard software metric should reach following requirements [1]:

- It can be calculated, uniquely, for all programs to which we apply it.
- It doesn't need to be calculated for programs that change size dynamically or programs that, in principle, cannot be debugged
- Adding something to a program (e.g., instructions, storage, processing time) will never decrease the measured complexity.

The first requirement assures a usable and objective measure. The second guarantees that the metrics are applied to the reasonable programs and the third is the intuitive understanding.

2.2 Software Metrics Used in This Study

The following items are brief description of the three data sets used in this research.

- *Large Legacy Telecommunications System (LLTS)*: LLTS consists of four releases with a large amount of software metrics, approximately 3500 to 4000 modules from several million lines of code in each release.
- *Network Telecommunications (NT)*: NT was collected from a network telecommunication systems, which has about 1.3 million lines of code. It is written in a high level programming language, PASCAL.

- *Large Network Telecommunications System (LNTS)*: LNTS was collected from another large network telecommunications system. It has about 13 million lines of code, written by another high level programming language, PROTEL.

2.3 LLTS metrics

Metrics for the Large Legacy Telecommunications System (LLTS) were collected using the Enhanced Measurement for Early Risk Assessment of Latent Defects (EMERALD) system, a decision support system that facilitates software measurement and software quality models [7]. EMERALD utilizes the Datrix software analyzer developed by Bell Canada [21] to measure the static attributes of the source code. The measurement by EMERALD is done periodically on the latest source code. The basic unit that Datrix measures is a procedure: a function or subroutine. Then, EMERALD gathers metrics that Datrix provides from the module level. We classified the metric collected from EMERALD into 3 classes as mentioned. Table 2.1 lists the names and descriptions of twenty-four product metrics of LLTS data. Table 2.2 and Table 2.3 lists fourteen process metrics and four execution metrics of LLTS data respectively [15].

2.3.1 LLTS product Metrics

Product metrics for LLTS can be further organized into three main groups: call graph metrics, control flow graph metrics and statement metrics. Call graph metrics give the call relation among the procedures. Control flow graph metrics

Table 2.1: Software Product Metrics of LLTS data

Symbol	Description
Call Graph Metrics	
<i>CALUNQ</i>	Number of distinct procedure calls to others.
<i>CAL2</i>	Number of second and following calls to others. $CAL2 = CAL - CALUNQ$ where <i>CAL</i> is the total number of calls.
Control Flow Graph Metrics	
<i>CNDNOT</i>	Number of arcs that are not conditional arcs.
<i>IFTH</i>	Number of non-loop conditional arcs (i.e., if-then constructs).
<i>LOP</i>	Number of loop constructs.
<i>CNDSPNSM</i>	Total span of branches of conditional arcs. The unit of measure is arcs.
<i>CNDSPNMX</i>	Maximum span of branches of conditional arcs.
<i>CTRNSTMX</i>	Maximum control structure nesting.
<i>KNT</i>	Number of knots. A “knot” in a control flow graph is where arcs cross due to a violation of structured programming principles.
<i>NDSINT</i>	Number of internal nodes (i.e., not an entry, exit, or pending node).
<i>NDSENT</i>	Number of entry nodes.
<i>NDSEXT</i>	Number of exit nodes.
<i>NDSPND</i>	Number of pending nodes (i.e., dead code segments).
<i>LGPATH</i>	Base 2 logarithm of the number of independent paths.
Statement Metrics	
<i>FILINCUIQ</i>	Number of distinct include files.
<i>LOC</i>	Number of lines of code.
<i>STMCTL</i>	Number of control statements.
<i>STMDEC</i>	Number of declarative statements.
<i>STMEXE</i>	Number of executable statements.
<i>VARGLBUS</i>	Number of global variables used.
<i>VARSPNSM</i>	Total span of variables.
<i>VARSPNMX</i>	Maximum span of variables.
<i>VARUSDUQ</i>	Number of distinct variables used.
<i>VARUSD2</i>	Number of second and following uses of variables. $VARUSD2 = VARUSD - VARUSDUQ$ where <i>VARUSD</i> is the total number of variable uses.

Table 2.2: Software Process Metrics of LLTS data

Symbol	Description
<i>DES_PR</i>	Number of problems found by designers.
<i>BETA_PR</i>	Number of problems found during beta testing.
<i>DES_FIX</i>	Number of problems fixed that were found by designers.
<i>BETA_FIX</i>	Number of problems fixed that were found by beta testing in the prior release.
<i>CUST_FIX</i>	Number of problems fixed that were found by customers in the prior release.
<i>REQ_UPD</i>	Number of changes to the code due to new requirements.
<i>TOT_UPD</i>	Total number of changes to the code for any reason.
<i>REQ</i>	Number of distinct requirements that caused changes to the module.
<i>SRC_GRO</i>	Net increase in lines of code.
<i>SRC_MOD</i>	Net new and changed lines of code.
<i>UNQ_DES</i>	Number of different designers making changes.
<i>VLO_UPD</i>	Number of updates to this module by designers who had 10 or less total updates in entire company career.
<i>LO_UPD</i>	Number of updates to this module by designers who had between 11 and 20 total updates in entire company career.
<i>UPD_CAR</i>	Number of updates that designers had in their company careers.

Table 2.3: Software Execution Metrics of LLTS data

Symbol	Description
<i>USAGE</i>	Deployment percentage of the module.
<i>RESCPU</i>	Execution time (microseconds) of an average transaction on a system serving consumers.
<i>BUSCPU</i>	Execution time (microseconds) of an average transaction on a system serving businesses.
<i>TANCPU</i>	Execution time (microseconds) of an average transaction on a tandem system.

indicates the flow of control from one statement to another in the software system. Statement metrics measure the properties of the program text, regardless of the meaning of the text or the order of the text in the program [5].

Call graph is a directed graph that represents the trace of the procedures in a program module. A call graph gives a layout of the procedure and subroutine calls. It ignores details and shows the abstract model of software design. In general, the main procedure is denoted by a root node. A call is represented by an edge, and child nodes represent the called procedure or subroutines. Some examples of metrics collected from call graphs are the number of distinct procedure calls, CALUNQ, and the number of second and following calls, CAL2. These metrics can be collected in the early high level design phase of the development process.

Control Flow Graph is the directed graph that shows the flow of control from one statement to another. Its structure consists of a start node, a stop node, and the inner nodes lying on several paths. Examples of control flow graph metrics are number of nodes and number of arcs in the graph, the out-degree and in-degree of a node, or conditional and unconditional arcs that create paths. Control flow graph metrics can be measured when the detailed design of the algorithm is completed.

Statement metrics can be collected when the program is in the non executing state. These metrics measure some properties of the source code without any interpretation. They only provide the quantity of a particular feature of the source

code. Therefore, this metrics will not be changed when the text is rearranged. Examples of statement metrics are lines of code (LOC), number of statements, number of unique operators and operands, total number of keyword appearances and total number of tokens [5]. For LOC, the treatment blank line and comment has to be described. In this study, we exclude blank lines, but includes comment lines.

2.3.2 LLTS Process Metrics

Process metrics give the detail of software-development activities. Prior research has shown that process metrics play a significant role in software quality prediction [2]. The discussion about process metrics can be divided in terms of internal and external process attributes as earlier mentioned. Internal attributes are the components that can be collected directly from the process development without executing the software, for example, the number of requirement reviews, number of requirement errors found during inspection, or even the number of programmers working on the project. For external attributes, they are only measured with respect to how the process relates to its environment. Examples of external attributes are cost, controllability and stability. Frequently, external attributes are measured in terms of internal attributes such as effectiveness of code maintenance, defined as the average number of faults discovered per thousand lines of code. Table 2.2 lists the available process metrics in LLTS data. However, process metrics are not applied in this study because the prediction of number of faults in a program was done early

in the software life cycle.

2.3.3 LLTS Execution Metrics

The execution metrics are collected from the real operation of the program in prior releases such as time or deployment. In this study, the execution metrics are USAGE and three time laboratory measurements, RESCPU, BUSCPU, and TANCPU. USAGE was calculated from deployment records in the previous releases [8]. The current releases are assumed to be similar to the prior ones because they were all deployed by the same users. Previous work [2] demonstrated that USAGE was a very important variable and played a significant role in the software quality model. Obviously, when the system has more users, The probability of users finding new faults is higher.

2.4 NT metrics

Network Telecommunications (NT) data is a network telecommunications system created by professional programmers. From 12 million lines of code, a large sample of modules, approximately 1.3 million lines of code, was used to collect software metrics. Table 2.4 gives the description of the NT system profile. NT metrics include lines of code, control-flow graph edges, and other non-declaration statements. There are 11 variables in the NT system metrics. For each module, nine static metrics were observed and recorded by using a software metrics analyzer package. Static metrics can be measured during the detailed design phase or later

Table 2.4: NT System Profile

Application	Telecommunications
Language	Pascal-like
Lines of Code	1.3 million
Executable Statements	1.0 million
Control Flow Graph Edges	364.0 thousand
Source Files	25.0 thousand
Functional Modules	2.0 thousand
Product Metrics	9
PCA Domain Metrics	5
Reuse Covariates	2
Quality Metric	Number of faults

from code. Besides these nine static metrics, NT metrics include two categorical variables, *ISNEW* and *ISCHG*.

$$ISNEW = \begin{cases} 1 & \text{a module did not exist in the prior release} \\ 0 & \text{otherwise} \end{cases}$$

$$ISCHG = \begin{cases} 1 & \text{a module was modified from prior release} \\ 0 & \text{otherwise} \end{cases}$$

The detail of NT product metrics are shown in table 2.5. For apparent understanding, McCabe cyclomatic complexity is calculated from

$$VG = Arcs - Vertices + Entrypoints + Exitpoints.$$

2.5 LNTS metrics

Large Network Telecommunications System (LNTS) metrics were collected using the Datrix software analyzer mentioned earlier in the EMERALD project. The

Table 2.5: NT Product Metrics

Symbol	Description
Call Graph Metrics	
<i>MU</i>	Number of modules used.
<i>TC</i>	Total calls to other modules.
<i>UC</i>	Unique calls to other modules.
Control Flow Graph Metrics	
<i>IFTH</i>	If-Then conditional arcs.
<i>LP</i>	Number of loops.
<i>NL</i>	Nesting level.
<i>SPC</i>	Span of conditional arcs.
<i>SPL</i>	Span of loops.
<i>VG</i>	McCabe cyclomatic complexity.

system profile of LNTS data is summarized in table 2.6 . Starting with 13 million lines of code, LNTS metrics were created from a subset of the entire software system. This subset covered only the modules modified during the development process. For the rest, modules were considered reliable because they remained unchanged. The subset consists of about 7 million lines of code, 180000 thousand source files, and approximately 7000 modified modules. The metrics structure is similar to NT metrics structure. The detail of LNTS product metrics can be found in table 2.7.

Table 2.6: LNTS System Profile

Application	Telecommunications
Language	PROTEL
Lines of Code	7.0 million
Executable Statements	6.0 million
Control Flow Graph Edges	2.0 million
Source Files	18.0 thousand
Changed Modules	7.0 thousand
Design Metrics	9
<i>PCA</i> Domain Metrics	4
Quality Metric	Number of faults

Table 2.7: LNTS Product Metrics

Symbol	Description
Call Graph Metrics	
<i>UCT</i>	Unique procedure calls
<i>TCT</i>	Total calls to others
<i>NDI</i>	Distinct files included
Control Flow Graph Metrics	
<i>VG</i>	McCabe's cyclomatic complexity
<i>NL</i>	Number of loops
<i>IFTH</i>	Number of if-then structures
<i>NELTOT</i>	Total nesting level
<i>PSCTOT</i>	Total number of vertices within the span of loops or if-then structures
<i>RLSTOT</i>	Total edges plus vertices within loop structures

Chapter 3

METHODOLOGY

This chapter discusses Module-Order Modeling (MOM) in detail, and the methodologies used in this study. It also gives an overview of principal components analysis (PCA), underlying quantitative models and how to use these methodologies in our research.

3.1 Module-Order Modeling

Software quality modeling is mainly used to predict the quality of modules to perform cost-effective enhancements before the software becomes operational. However, defining such modules to be fault-prone requires a specific threshold before modeling. This is frequently not suitable because of the undetermined amount of reliability enhancement effort. Therefore, predicting the rank-order of modules is often more useful. The module-order modeling ranks the modules according to the predicted quality factor, and gives management more flexible reliability enhancement strategies than classification models [12].

Module-order model uses the product and process metrics to predict the rank-order of quality factor such as number of faults or code churn. The goal is to develop a suitable model that predicts the ranking of modules, according to a quality factor from the most to the least fault-prone. An efficient model can help the engineers to focus on, and enhance the modules that possibly bring the greatest payoff when the product is released.

A module-order model consists of the following components [12].

1. An underlying quantitative software quality prediction model
2. A ranking of modules according to a quality measure predicted by the underlying model
3. A procedure for evaluating the accuracy of a model's ranking

In this study, the underlying quantitative software quality models are Case-Based Reasoning, Multiple Linear Regression, Artificial Neural Network, SPLUS and CART algorithms for tree modeling. All techniques use different means to predict the dependent variables. The inputs to these modeling methods are the process and product metrics collected during the development process.

There are two kinds of input data sets provided to the underlying quantitative models: *fit* and *test* data sets . In general, we use the *fit* data set to build the models based on different techniques and use the *test* data set to evaluate the models. When evaluating the model, the accuracy of the result is measured by the difference

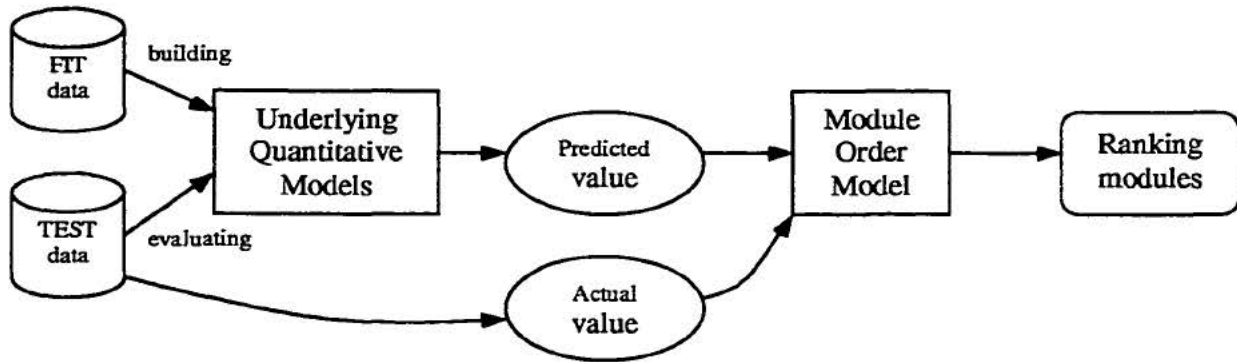


Figure 3.1: Module-order model operation

between actual and predicted dependent variables. Two statistical values, AAE and ARE, are used to represent the accuracy of the prediction. Small AAE and ARE values indicate an accurate model.

The input to the module-order models is the actual quality factor (dependent variable) measured during the development process and the predicted quality factor. The output of the module-order model is the ranking of the modules based on the predicted quality factor of each module. Our goal is to compare the performance of module-order models based on different underlying quantitative models and relate the performance of MOM to the accuracy of the underlying models. In this study module-order modeling is applied to the *test* data set . The concise scope of module-order model operation is shown in Figure 3.1

Since the quality factor we are interested in this study is the number of faults, the predicted quality factor must have at least an ordinal scale (1,2,3,...) [18]. In

contrast, the quality factor of classification models has a nominal scale, fault-prone or not fault-prone modules.

Building the software quality model uses the following steps [26].

1. Build a model using data from past projects: The *fit* data set is used to build the model. In some methodologies, many models can be built by varying the main parameter in the algorithm. The model, which has the best accuracy will be chosen.
2. Evaluate the model using distinguished historical data: After building the model, the *test* data set is used to evaluate the model and test the quality of fit.
3. Use the model on the current project's data: The acceptable model is applied to the actual project's data we want the dependent variable predicted for.

According to the modeling technique, the underlying quantitative model is built. Suppose a quantitative model has an actual dependent variable represented by F_i , and $f(x_i)$ is an ideal function of independent variables, the vector x_i .

$$F_i = f(x_i) \quad (3.1)$$

Then, suppose $\hat{F}(x_i)$ be the predicted dependent variable of F_i estimated by a fitted quantitative model, $\hat{f}(x_i)$.

$$\hat{F}(x_i) = \hat{f}(x_i) \quad (3.2)$$

All independent variables are chosen by the selection method included in the modeling technique.

Spearman correlation [24] was proposed to evaluate the accuracy between actual and predicted ranking of a module-order model. It represents the overall ranking accuracy over the entire data set, but it does not give a measure of robustness. In our case, we do not care about the accuracy over the entire data set, but we want to see the performance of a module-order model in terms of robustness (explained later in this section). Therefore, Spearman correlation is not appropriated for evaluation of a module-order model [10].

For module-order modeling, the accuracy of the predicted dependent variable is not the goal. However, we concentrate on the ability of the model to approximate an ordering starting with the most fault-prone to a certain percentile. Therefore, we use the following method to evaluate how a module-order model performs.

Let R_i be the percentile rank of observation i in a perfect ranking, \mathbf{R} . Let $\hat{R}(x_i)$ be the percentile rank of observation i in a predicted ranking, $\hat{\mathbf{R}}$.

1. Determine the perfect ranking of modules in the test data set, \mathbf{R} , by ordering modules according to F_i (actual software quality factor).
2. Determine the predicted ranking, $\hat{\mathbf{R}}$, by ordering modules according to $\hat{F}(x_i)$ (predicted software quality factor) from the least to the most fault-prone.

3. After applying module-order modeling, modules are ordered for reliability improvement beginning with the most fault-prone module and ending with the least fault-prone modules at the highest considered percentage. A cutoff point, c , is the percentile indicating the last module included in the enhancement process.

Management will choose to enhance the module in priority order according to the ranking. However, since the rank of the last module enhanced is unknown at the time of modeling, we choose a range of percentiles, C , that might be in the interest of the manager to consider. In this research, we covered the 50 percent of the entire modules. In general, there is no project where more than 50 percent of the modules will be reviewed. Other projects may choose different percentiles.

For each cutoff percentile value of interest, $c \in C$:

- Calculate the sum of actual number of faults, $G(c)$, in modules above the cutoff percentile for perfect ranking, \mathbf{R}

$$G(c) = \sum_{i:R_i \geq c} F_i \quad (3.3)$$

- Calculate the sum of actual number of faults in modules above the cutoff percentile $\hat{G}(c)$ for predicted ranking, $\hat{\mathbf{R}}$

$$\hat{G}(c) = \sum_{i:\hat{R}(x_i) \geq c} F_i \quad (3.4)$$

4. Let G_{tot} be the total number of actual faults. Calculate the percentage of faults of two rankings rationalized by the G_{tot} , $G(c)/G_{tot}$ and \hat{G}/G_{tot} . This value shows us the benefit of using the model at the cutoff point, c
5. Calculate the performance of the model, $\phi(c)$, representing how closely the faults in predicted ranking match those in the perfect ranking.

$$\phi(c) = \frac{\hat{G}(c)}{G(c)} \quad (3.5)$$

Higher c implies the more fault-prone modules.

The ratio of the actual faults at c , $\phi(c)$, determines the performance of the model's ranking at the given cutoff c compared to the perfect ranking. The variation of $\phi(c)$ shows the robustness of the model. If the variation of $\phi(c)$ over the range C is small, it refers that the model is robust. Due to the uncertain resources for reliability enhancement, we prefer to see the consistency of the accuracy over a range of c .

All modules above the selected cutoff point would be considered to be reviewed in the same way, and those below the cutoff point would be considered to be acceptable. Therefore, the difference between the model's ranking and perfect ranking is not a suitable measure of the model's accuracy. The accuracy of the rank-order within the enhanced group (above the selected c), nor within the non-enhanced group (below the selected c) are relevant. However, the high

accuracy, $\phi(c)$, at the selected cutoff point c is preferred because the model's accuracy at the cutoff point c is only interested for management view.

3.2 Classification

We can also use module-order modeling to classify modules [11]. The modules above the cutoff percentile c are considered as fault-prone, and the other below are considered as not fault-prone.

Classification can be performed when the threshold value is determined. This number depends on the project specification. Different projects have different threshold. When the threshold number is given, we can classify the modules. The accuracy of classification is measured by means of misclassification rates. As in general, the same mathematical method for classification is applied to module-order model. The defined threshold is translated to a corresponding c cutoff percentile of modules. Type I misclassification is when we specify the module to be fault-prone, but it's actually not fault-prone. On the other hand, Type II misclassification is when the module is determined to be not fault-prone, but it's actually fault-prone. Obviously, Type II misclassification is more severe than Type I because the actual fault-prone modules are not reviewed, and are more expensive to fix when faults are found by the customer [12].

In management aspect, misclassification rates are not appropriate. Consequently, we use effectiveness and efficiency, which are more relevant to management

view [10, 9]. Effectiveness is defined as the proportion of fault-prone modules correctly identified. If we review a not fault-prone module, we waste the time because the module is already in an acceptable condition. Efficiency is the proportion of reliability enhancement effort that is not wasted. Effectiveness is maximized by minimizing Type II misclassification, and efficiency is maximized by minimizing Type I misclassification. This shows that effectiveness and efficiency have the same variation as Type I and Type II misclassification. When the one increases, the other decreases. It is the same tradeoff as effectiveness and efficiency. Further details are shown in Table 3.1 .

3.3 SMART

The section briefly introduces SMART, the Software Measurement Analysis and Reliability Toolkit, and how we used SMART in our study.

Currently, SMART handles four types of models, case-based reasoning (CBR), CBR with two data clustering, CBR with three data clustering and module-order model [13]. Since our study concentrates on module-order model (MOM), we will only explain the feature of MOM in SMART.

SMART architecture can be grouped into 3 main parts shown in Figure 3.2.

- **Data manager:** This part operates the input data, *fit* and *test* data set. *Fit* data set is used to build the model, and *test* data set is used to validate the model.

Table 3.1: Effectiveness and efficiency

G_1	<i>Not fault-prone</i> group (class)
G_2	<i>Fault-prone</i> group (class)
$Class_i$	Actual class of module i
$Class(x_i)$	Predicted class of module i based on vector of independent variable, x_i
π_1	Expected proportion of <i>not fault-prone</i> modules
π_2	Expected proportion of <i>fault-prone</i> modules
$Pr\{1 1\}$	Rate of correct classifications of <i>not fault-prone</i> modules
	$Pr\{1 1\} = Pr\{Class(x_i) = G_1 Class_i = G_1\}$
$Pr\{2 2\}$	Rate of correct classifications of <i>fault-prone</i> modules
$Pr\{2 1\}$	Type I misclassification rate
$Pr\{1 2\}$	Type II misclassification rate
<i>effectiveness</i>	Proportion of <i>fault-prone</i> modules that received reliability enhancement treatment out of all the <i>fault-prone</i> modules
	$effectiveness = Pr\{2 2\} = 1 - Pr\{1 2\}$
<i>efficiency</i>	Proportion of <i>fault-prone</i> modules that received reliability enhancement treatment out of all modules that received it.
	$efficiency = \frac{Pr\{2 2\}\pi_2}{Pr\{2 1\}\pi_1 + Pr\{2 2\}\pi_2}$

- **User Interface:** Using dialog-based property sheet, graphical interface helps the user to control the tool comfortably.
- **Data analysis:** Using the input from data manager, four types of models are chosen to analyze the data according to the user's purpose.

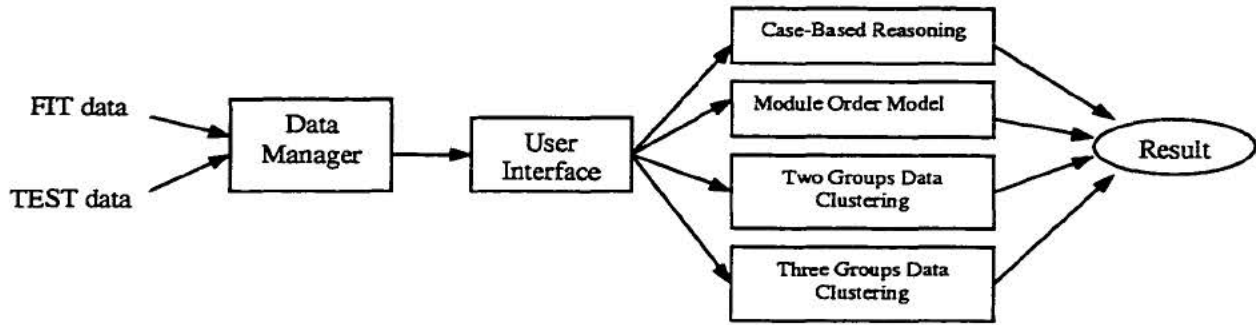


Figure 3.2: Smart Architecture

Usually, module-order model in SMART is designed to have multiple linear regression (MLR) as underlying quantitative models. The weights used in this technique are provided by the user. When the input data set is entered, MLR automatically performs its operations and estimates the predicted dependent variable. After that MOM can build the model by using that given variables.

Based on the contribution of this research, the goal is to study the performance of MOM based on different underlying quantitative software models. Therefore, we input predicted variables from the different techniques we previously mentioned. In the latest updated version, MOM allows the user to input a set of predicted variables obtained from other modeling techniques besides MLR in the “Using Input Prediction” console as shown in the Figure 3.3.

The statistical result from MOM are labelled C1 through C8.

- C1: Percentage of $G(c)$ based on sum of dependent values in all modules
- C2: Percentage of $\hat{G}(c)$ based on sum of dependent values in all modules

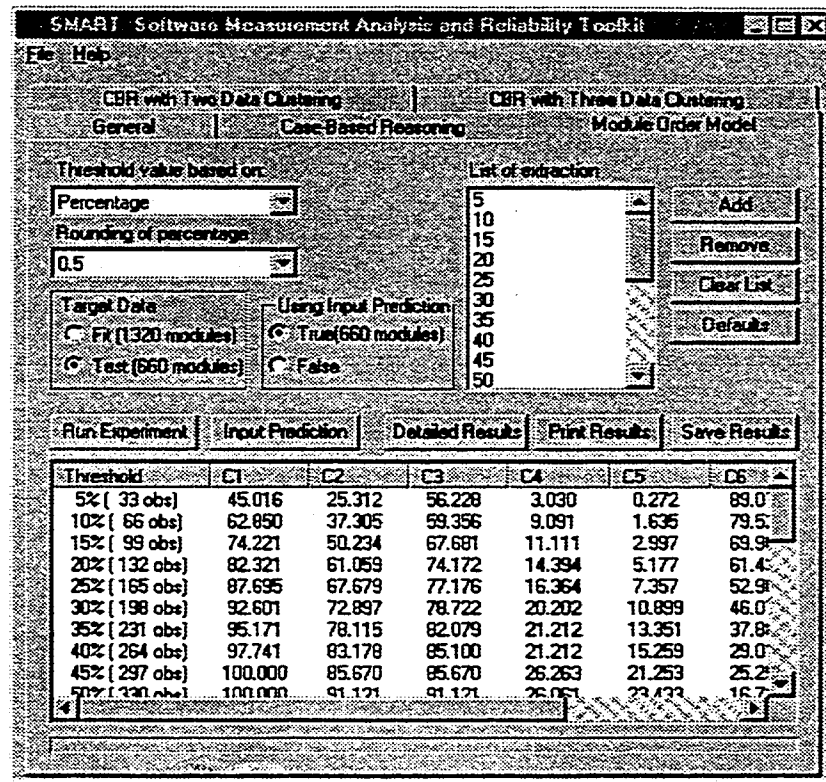


Figure 3.3: MOM page in SMART

- C3: Percentage of the model's accuracy at the given cutoff, $\phi(c) : C1/C2$
- C4: Model's inefficiency, 1-efficiency
- C5: Type I misclassification rate at the specific cutoff
- C6: Type II misclassification rate at the specific cutoff
- C7: Model's effectiveness
- C8: Model's efficiency

where, $G(c)$ is sum of the actual dependent variable values above the cutoff under a perfect ranking, and $\hat{G}(c)$ is the sum of actual dependent variable values above the cutoff under the predicted ranking.

3.4 Principal Components Analysis

Most independent variables have a high correlation among each other. This causes degradation of software quality models because a slight change in the fit data set makes the models very unstable. Principal components analysis is used to solve this problem.

Principal components analysis is the technique used to remove the correlation among all independent variables. It transforms the raw data set into principal component data set, reducing the number of variables, but without losing the significant variation. In principal components form, the independent variables are not correlated. They are a set of orthogonal vectors.

Consider a data set that has n modules and each module has m independent variables. This produces the metric $n \times m$ dimensions where all variables in all columns are standardized, having a mean of zero and a variance of one. We called this \mathbf{Z} metric. The principal components are the linear combinations of m standardized random variables, Z_1, \dots, Z_m . Then, proceeding the following steps will obtain the principal components.

1. Calculate the covariance matrix, Σ , of \mathbf{Z}

2. Calculate the eigenvalues, λ_j , and eigenvectors, \mathbf{e}_j , $j = 1, \dots, m$. Each eigenvalue is the variance of the corresponding principal component.
3. Because the eigenvalues series are decreasing, $\lambda_1 \geq \dots \geq \lambda_m$, the dimensionality of the data can be reduced without losing the significant variance by considering only the first p components, $p \ll m$.

We want to achieve at least 90% of variance of the original standardized metrics, so we choose the minimum p such that $\sum_{j=1}^p \lambda_j/m \geq 0.90$.

4. Calculate the $m \times p$ standardized transformation matrix, \mathbf{T} , whose each column, \mathbf{t}_j , is defined as

$$\mathbf{t}_j = \frac{\mathbf{e}_j}{\sqrt{\lambda_j}} \text{ for } j = 1, \dots, p \quad (3.6)$$

5. Calculate domain metrics, D_j , for each module. \mathbf{D} is an $n \times p$ matrix with D_j values for each column, $j = 1, \dots, p$.

$$D_j = \mathbf{Z}\mathbf{t}_j \quad (3.7)$$

$$\mathbf{D} = \mathbf{Z}\mathbf{T} \quad (3.8)$$

The principal component variables are uncorrelated and suitable to build software quality models. Each component has a mean of zero and a variance of one.

3.5 Model Performance Evaluation

The accuracy of quantitative software quality models can be evaluated by calculating the error values. Two common statistics for evaluating predictions are average absolute error, AAE, and average relative error, ARE.

$$AAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (3.9)$$

$$ARE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i + 1} \right| \quad (3.10)$$

In the equation, n is the number of modules in the data set. y_i is the actual dependent variable and \hat{y}_i is the predicted dependent variable from the quantitative model. For ARE, the denominator has one added to avoid division by zero [17].

3.6 Underlying quantitative models

In this study, we input the predicted value from 5 quantitative prediction models to a module-order model (MOM) and observe the performance of MOM. We want to compare the result of MOM based on different quantitative models. This section covers the brief description of these 5 methodologies.

3.6.1 Case-Based Reasoning

Case-Based Reasoning, CBR, is the technique for predicting the software quality factors by using the historical data [29]. Thus the data stored in the database are the *cases* in a *case library*. Applying CBR to the system, when the new data is

obtained, the system measures the difference between the new data and the *cases*. The algorithm chooses the most similar cases and generates the solution to the new input data.

The difference between the retrieved data and cases is measured in the form of a “distance”. To compute an efficient similarity (or distance), several similarity functions are used.

- **Case Similarity Function**

Suppose the test data set is the $i \times k$ dimensional matrix. Let x_i be the vector of metrics of the i^{th} module (row) from the new input, test data set, and there are k metrics (column) in each module. x_{ik} represents the k^{th} component in the i^{th} module. For the data in the case library, suppose the case or fit data set is the $j \times k$ dimensional matrix. Let c_j be the vector of the j^{th} module (row) in the case, fit data set. The case library has the same number of metrics for each module then the test data, so c_{jk} stands for the k^{th} (column) component in the j^{th} module.

- **Euclidean Distance**

$$d_{ij} = \left(\sum_{k=1}^m (w_j (c_{jk} - x_{ik}))^2 \right)^{\frac{1}{2}} \quad (3.11)$$

where, m is the number of variables and w_j is weighted in each j^{th} variable, approved by the user.

– **Absolute Distance**

$$d_{ij} = \sum_{k=1}^m w_k |c_{jk} - x_{ik}| \quad (3.12)$$

– **Mahalanobis Distance**

This algorithm is used when the variables in the data set are highly correlated.

$$d_{ij} = (\mathbf{c}_j - \mathbf{x}_i)' \mathbf{S}^{-1} (\mathbf{c}_j - \mathbf{x}_i) \quad (3.13)$$

\mathbf{S} is the covariance matrix of the variables for all modules in the case library. \mathbf{S}^{-1} is the inverse of \mathbf{S} . Prime ($'$) means transpose.

The smaller distance presents the more similarity of new input and cases. After the most similar cases are selected, the system determines the answer by using a solution algorithm.

• **Solution Algorithm**

The most similar cases are represented by the *nearest neighbors*. Let N be the complete set of *nearest neighbors*, which are the most similar cases in the fit data set to the case in the test data set. The number of *nearest neighbors*, n_N is defined by the user.

– **Unweighted Average**

$$\hat{y}_i = \frac{1}{n_N} \sum_{j \in N} y_j \quad (3.14)$$

where \hat{y}_i is the predicted dependent variable and y_j is an actual dependent variable from module j in the case library.

– **Weighted Average**

$$\delta_{ij} = \frac{1/d_{ij}}{\sum_{j \in N} 1/d_{ij}} \quad (3.15)$$

$$\hat{y}_i = \sum_{j \in N} \delta_{ij} y_j \quad (3.16)$$

The output of solution algorithm is the predicted result created by Case-Based Reasoning methodology. More detail in CBR approach can be found in [29].

3.6.2 Multiple Linear Regression

Multiple Linear Regression is one of the instrument widely used to define the dependent variable by using a statistical function, formed by the known independent variables [12]. It has the following general form.

$$\hat{y}_i = a_0 + a_1 x_{i1} + \dots + a_p x_{ip} \quad (3.17)$$

$$y_i = a_0 + a_1 x_{i1} + \dots + a_p x_{ip} + e_i \quad (3.18)$$

Where \hat{y}_i is the predicted value of the i^{th} observation, y_i is the actual value of dependent variable, and $e_i = y_i - \hat{y}_i$ is the error of i^{th} observation. x_{i1}, \dots, x_{ip} are the independent variables and a_0, \dots, a_p are estimated parameter, calculated by the least squares method. This method has for criteria to choose a group of number that minimize $\sum_{i=1}^n e_i^2$ [23].

Since the independent variables play an important role to build the model, they have to be preprocessed to remove any correlation and some insignificant variables. These insignificant variables may cause interpretation to be inaccurate if they are added to the model. Therefore, we need to choose only the significant variables to be included in the model. The process used to determine the significant variables is called *model selection*.

Among available model selection techniques, we use the stepwise regression method. Stepwise regression is an iterative process. In each round, the process either adds or removes variables from the model, based on the significant level of α in F test [29].

3.6.3 Artificial Neural Network

Artificial Neural Network (ANN) applies the simulation of the organizational process in the human brain to compute the output when the input is provided [29]. It can be classified into 2 groups, supervised-learning and unsupervised-learning network based on learning rules [20]. In this research, we studied supervised learning network.

When giving input to the system, supervised learning network responds with the desired output at the instance of time. The network automatically realizes what output should come out. We focus our study to feed forward and back propagation supervised-learning neural networks.

Neural networks consists of neurons. For feed forward model, suppose we have x_i input and k^{th} processing elements. The elements compute sum of the weights multiplied by its input, x_j and basis, b_k . The result of this computation is the input to activation function, $f(\cdot)$. The output of the activation function, o_k , is the output of the neuron and the answer of the network (dependent variable) [29]. The operation of the neuron can be described in the following equation.

$$net_k = w_{1k}x_1 + w_{2k}x_2 + \dots + w_{mk}x_m + b_k \quad (3.19)$$

$$o_k = f(net_k) \quad (3.20)$$

where, m is the number of inputs (independent variables).

For back propagation model [19, 25, 31], the system initializes the process with the set of random parameters, weights and basis. Training is required to adjust these parameters [29]. At the time of training, a set of input-output pairs are entered. When input is propagated through the network, the network calculates the weighted sum of input vector and basis, and finally comes up with the output from the activation function as for the feed forward model. Then, the output of the network is compared to the expected output of the input-output pairs, and the difference (error vector) is propagated back to train the network to minimize the error and find the optimum weights. The training stops only when the squared error satisfies the setting point

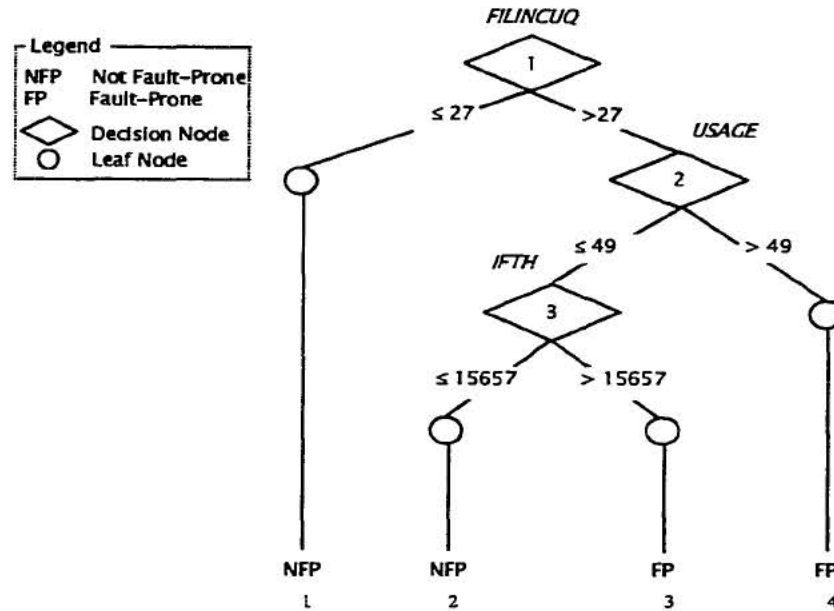


Figure 3.4: Example of Tree model in purpose of classification

3.6.4 Tree model

Tree model is the exploratory technique that gives the result displayed in the form of tree based on decision rules. Beginning at the root node, the algorithm splits a downward path in the trees, one node after another, until it reaches a leaf node. Each node represents independent variables, each edge represents a possible result of the decision, and the leaf node represents the final answer of either classification or regression as an example in Figure 3.4. The parameter making the decision in each node is called a *predictor*.

While building a Tree model, the data set is splitting continuously until it reaches the point based on stop-splitting rules. The tree is binary if the parent node

always split into exactly two child nodes associated with the decision and the child node is repeatedly considered as parent, then recursively behave the same way. All tree models in this research are binary trees.

For this study, we focus on regression. The output from the tree is the predicted value for each module in the data set. Of several available tree methodologies, we use SPLUS and CART algorithm. According to binary recursive partitioning definition, both approaches generate binary trees, but they have different algorithms and rules. More details are briefly discussed in the next section.

3.6.5 CART

CART is a statistical tool providing the algorithm to generate a tree model. Cart algorithm partitions the data into binary paths until reaching the terminal or leaf node based on two rules, least square (LS) and least absolute deviation method (LAD) [27].

- Least Squares Method (LS): This method uses the mean value computed in a particular node as the predicted value in that node, and use mean-square error as the standard for considering the goodness-of-split.
- Least Absolute Deviation (LAD): This method uses the median value computed in a node as the predicted value in that node, and mean absolute error is used as the rule for measuring the goodness-of-split.

The goodness-of-split is the criterion used to control the construction of the tree. Since we are interested in obtaining the predicted value, we will focus only on regression tree and ignore classification feature of tree model. For cart, when generating the tree, the descendant node is more homogeneous (purer) than its parent node. Purity of the tree is determined in terms of variance, LS or LAD up to the selected feature. The goal is to minimize the LS or LAD value in splitting nodes. Purity condition (goodness of split) of the node is calculated after and before split, and purity should be increased after the split. The split stops when there is no diversity of variance in the terminal node [27].

3.6.6 SPLUS

SPLUS is another statistical tool that can be utilized to build tree model for prediction purpose [27]. SPLUS algorithm grows the tree based on two core factors, *minsize* and *mindev*. The input to this algorithm is a set of independent variables and dependent variables responding in each observation.

- *minsize*: Abbreviated from minimum size, this value is the threshold to limit number of observations in the leaf node. If the number of observations in a node less than *minsize*, the tree stops growing.
- *mindev*: Abbreviated from minimum deviance, this value is also the threshold to limit the growth of the tree. If the deviance in a node is less than *mindev*, the tree stops growing. Then, that node becomes a leaf node.

Suppose y_i be the dependent value for observation i , $\mu(S)$ be the mean of dependent variables over the data set S , and $|S|$ be the number of observations in the data set S . The deviance of i^{th} observation is defined as

$$D(S) = (y_i - \mu_S)^2 \quad (3.21)$$

The algorithm chooses the predictor that maximize the change in deviance, and The cutoff value of the chosen predictor is selected based on minimizing the sum of deviances of the left(L) and right(R) child nodes: $D(S_L) + D(S_R)$

Chapter 4

EXPERIMENTS

This chapter describes the experiment conducted in this study. The results obtained from the underlying quantitative models are tabulated. We use graphical presentation to show the performance of the module-order models.

4.1 Case Study Methodology

Before the module-order modeling is performed, we need to retrieve the predicted values of the quality factor from the underlying quantitative models. This section summarizes our methodology to build and validate the underlying quantitative models in our case studies. Case studies are based on past development projects.

1. Preprocess measurements: The raw software metrics may not be suitable as inputs because of some insufficient attributes such as the variety of unit measurements or correlation among data. These properties degrade the interpretation of the model. Standardizing the data to have a mean of zero and a variance of one for each metric provides a single unit measurement for the data. In

addition, performing principal components analysis on the standardized data removes the correlation among the metrics. In this study, we use two types of data sets, PCA is the set of software metrics on which we performed principal component analysis, and RAW is the original software metrics collected during the software development process.

2. Choose a model validation strategy: The following strategies are used to define the fit and test data sets for our three case studies: data splitting [4] for NT and LNTS data, and subsequent releases [4] for LLTS data.
3. Prepare fit and test data sets: After choosing the validation strategy, we created the fit and test data sets for the three case studies. In this study, the dependent variable is the number of faults. The independent variables are software product and process metrics.
4. Select significant variables: Several independent variables are measured during the development process. This process removes the insignificant variables that can degrade the interpretation of the model.
5. Build the model based on the underlying quantitative models: Use the operational fit data set to build the models based on different underlying techniques.
6. Evaluate the model: Apply the test data set to the built model and determine the accuracy of each underlying methods. We used two statistical indicators,

AAE and ARE, to measure the accuracy.

After completing this process, we derive the predicted dependent variable and compute the prediction accuracy for each different underlying quantitative model. This predicted metric is provided to a module-order model to build a ranking. The detail of the experiments including results from all three case studies are described in the next section.

4.2 System Description

This section fully describes the three data sets used in the experiments. Each case study has two forms of data, RAW and PCA. Specific details are thoroughly explained for each software systems.

4.2.1 LLTS System Description

The first experiment was applied to LLTS metrics. As stated in chapter 2, LLTS (Large Legacy Telecommunication Systems) metrics were collected using EMERALD. The system was written in a high level programming language. The LLTS system metrics comprises four releases of data accumulated over past projects. We refer to them as releases 1, 2, 3 and 4. Each release has a different number of observations. They are 3649, 3981, 3541 and 3978 observations respectively in each release. An observation is associated with a module of source-code files in the software system. A release of LLTS metrics consists of 42 process, product and

execution metrics. In this study, we used the 24 product metrics and 4 execution metrics (28 raw metrics) as independent variables. The dependent variable was the number of faults inspected in a module during the past testing. The number of faults was calculated from the sum of three metrics: *CUST_PR*, *DES_PR* and *BETA_PR*. These three metrics are the number of faults detected by customers, by designers and during the beta testing period respectively. We refer to this data set of metrics as LLTS-RAW.

In addition to the 28 raw metrics, the metrics were transformed using principal components analysis (PCA) in order to reduce the correlation. Prior research [29] shows that the product, process and execution groups of metrics were not much correlated to each other. Therefore, principal components analysis process was only conducted for process and product metrics, and execution metrics were used without preprocessing. The detail of performing PCA on raw metrics is shown in Table 4.1. Six principal components were extracted from twenty-four product metrics. The table represents a 24×6 matrix, which 24 rows represent the 24 product metrics and the 6 columns represents six derived principal components. The values in the matrix show the correlation between raw metrics and principal components. Higher values indicate a strong correlation. If the value is one, it denotes that two metrics have the same meaning. This implies that the principal components can replace the raw metrics as example in Table 4.1.

In Table 4.1, the highest values in each row are highlighted in bold. From

Table 4.1: Factor Pattern for Principal Components of Product Metrics for LLTS data set

<i>Metric</i>	<i>PROD1</i>	<i>PROD2</i>	<i>PROD3</i>	<i>PROD4</i>	<i>PROD5</i>	<i>PROD6</i>
<i>CALUNQ</i>	0.90241	0.05180	0.10442	0.23226	0.17394	0.06161
<i>VARUSDUQ</i>	0.89496	0.18889	0.15268	0.17704	0.14681	0.19375
<i>LOC</i>	0.88610	0.28067	0.18160	0.16929	0.16431	0.14445
<i>NDSENT</i>	0.87966	-0.11142	0.01770	0.18394	0.10988	0.17201
<i>STMEXE</i>	0.86869	0.25870	0.17612	0.17324	0.26880	0.07169
<i>STMCTL</i>	0.86701	0.26070	0.27411	0.17258	0.08509	0.17429
<i>NDSEXT</i>	0.84668	0.01970	0.10855	0.20099	0.08568	0.35294
<i>STMDEC</i>	0.84595	0.20127	0.14148	0.14922	0.07117	0.14898
<i>IFTH</i>	0.84569	0.34158	0.27880	0.18162	0.10404	0.10659
<i>NDSINT</i>	0.84185	0.34355	0.27606	0.15248	0.18487	0.10920
<i>CNDNOT</i>	0.83478	0.31173	0.26233	0.15217	0.23697	0.17495
<i>LOP</i>	0.82816	0.10817	0.20842	0.01714	0.02129	-0.09590
<i>VARGLBUS</i>	0.80191	0.35962	0.20123	0.14369	0.21197	0.20453
<i>VARUSD2</i>	0.79088	0.44096	0.27108	0.11186	0.18082	0.12928
<i>CAL2</i>	0.59715	0.20418	0.07284	0.19317	0.56903	-0.05255
<i>VARSPNSM</i>	0.39174	0.86022	0.17718	0.10430	0.06747	0.08423
<i>VARSPNMX</i>	0.14039	0.83489	0.17722	0.35150	0.10357	0.09136
<i>CNDSPNMX</i>	0.12121	0.27629	0.75661	0.14289	0.25648	0.30600
<i>CTRNSTMX</i>	0.32233	0.09595	0.70922	0.42101	-0.00726	-0.01574
<i>CNDSPNSM</i>	0.60974	0.21553	0.64240	0.00704	0.22007	0.13087
<i>FILINCUCQ</i>	0.39561	0.25790	0.15541	0.72651	-0.03570	0.16963
<i>LGPATH</i>	0.21017	0.37957	0.35793	0.63962	0.16986	-0.04151
<i>KNT</i>	0.21362	0.06906	0.17464	-0.00640	0.88896	0.09719
<i>NDSPND</i>	0.40212	0.14886	0.21690	0.07507	0.08412	0.81557
Variance	11.61638	2.82091	2.37167	1.69515	1.64281	1.23002
% Var.	48.40%	11.75%	9.88%	7.06%	6.85%	5.13%
Cum. %	48.40%	60.15%	70.03%	77.09%	83.94%	89.07%

Stopping rule: at least 89% of variance

the table, PROD1 is highly correlated to fifteen metrics: CALUNQ, VARUSDUQ, LOC, NDSSENT, etc; PROD2 is highly correlated to VARSPNSM and VARSPNMX; PROD3 has high correlation with CNDSPAMX, CTRNSTMX and CNDSPNS, for instance. For LLTS data set, we used 89% of variance as stopping rule to generate principal components. When combining six principal components from 24 product metrics with 4 execution metrics, the 10 PCA metrics were generated. We refer to this data set as LLTS-PCA data set.

4.2.2 NT System Description

NT metrics are collected from the network telecommunication systems. The data splitting technique was used to generate the fit and test data sets. Two-third of the modules (1320 observations) of the original data are impartially split into a fit data set and the rest (660 observations) are used as a test data set. The 11 independent variables of NT include nine product metrics and two categorical variables (reuse covariates). The dependent variable is the number of faults found in software modules during the past testing phase. We refer to this 11 raw metrics as NT-RAW data set.

Principal components analysis was performed on the original nine product metrics. Three components were retrieved. The detail of the three principal components extracted from the nine metrics is shown in Table 4.2. The table presents a 9×3 matrix, which nine rows represent nine product metrics and three columns

Table 4.2: Factor Pattern for Principal Components of Design Product Metrics for NT data set

<i>Metric</i>	<i>DOMAIN1</i>	<i>DOMAIN2</i>	<i>DOMAIN3</i>
<i>SPL</i>	0.901	0.359	0.137
<i>LP</i>	0.880	0.370	0.134
<i>SPC</i>	0.719	0.545	0.316
<i>NL</i>	0.683	0.593	0.334
<i>TC</i>	0.359	0.864	0.216
<i>UC</i>	0.426	0.830	0.245
<i>VG</i>	0.597	0.724	0.309
<i>IFTH</i>	0.599	0.681	0.357
<i>MU</i>	0.177	0.265	0.939
Eigenvalues	3.630	3.410	1.460
% Variance	40.3%	37.9%	16.2%
Cumulative %	40.3%	78.1%	94.4%

Stopping rule: at least 94% of variance

stand for the three principal components, P1 , P2, and P3. The values denote the correlation between the raw metrics and principal components. We used 94.4% of the variance as the stopping rule to extract the principal components. Adding these three components with the 2 categorical variables, 5 independent variables for NT PCA data were created. We refer to this data as the NT-PCA data set.

4.2.3 LNTS System Description

LNTS metrics is the last data set used in this study. It was collected from a Large Network Telecommunication System. As for the NT data set, the data splitting technique was used to create the fit and test data sets. Two-third of the modules (4648 observations) was used as a fit data set and the remaining one-third

Table 4.3: Factor Pattern for Principal Components of Software Metrics for LNTS data set

Metric	<i>DOMAIN1</i>	<i>DOMAIN2</i>	<i>DOMAIN3</i>	<i>DOMAIN4</i>
<i>PSCTOT</i>	0.884	0.313	0.275	-0.009
<i>NELTOT</i>	0.853	0.362	0.335	0.012
<i>IFTH</i>	0.665	0.601	0.374	0.013
<i>TCT</i>	0.360	0.853	0.307	0.005
<i>UCT</i>	0.359	0.838	0.367	0.001
<i>VG</i>	0.617	0.632	0.416	-0.005
<i>NL</i>	0.290	0.407	0.841	-0.046
<i>RLSTOT</i>	0.418	0.316	0.827	-0.019
<i>NDI</i>	0.003	0.004	-0.030	0.999
Eigenvalues	2.85	2.69	2.12	1.00
% Variance	31.67%	29.89%	23.56%	11.11%
Cumulative %	31.67%	61.56%	85.12%	96.23%

Stopping rule: at least 96% of variance

(2324 observations) as a test data set. The independent variables for the raw data set comprise nine product metrics and the dependent variable is the number of faults found in the modules in the historical development. We refer to this data as LNTS-RAW data set in the remaining part of this thesis.

Principal components analysis was also conducted on the LNTS data set. The process extracted four principal components from the original nine product metrics. They are shown in Table 4.3. The table presents a 9×4 matrix, which nine rows represent nine product metrics and four columns represent four principal components, D1, D2, D3 and D4. The values in the matrix give the correlation between raw metrics and principal components. The principal components analysis was stopped when 96% of the variance was reached by the process. These four

principal components are referred to as LNTS-PCA data set.

4.3 Experiments on LLTS

The first experiment used the metrics from LLTS data set as input to the underlying quantitative models. This section discusses the result of the experiments conducted on the LLTS data sets, LLTS-RAW and LLTS-PCA. The experiment results are presented in the following order.

1. The prediction results of the five underlying quantitative models are shown in tabulated form. Those underlying quantitative models are Case-Based Reasoning (CBR), Multiple Linear Regression (MLR), Artificial Neural Network (ANN), Cart and SPLUS. For the Cart algorithm, the models were built based on two methodologies : Cart-Least square (Cart-LS) and Cart-Least Absolute Deviation (Cart-LAD) [27].
2. The module-order modeling results are displayed in graphs. Two types of graph are plotted to present the results.
 - Alberg diagram [24] showing the curve of the perfect ranking percentage ($G(c)/G_{tot}$) compared to the predicted ranking percentage ($\hat{G}(c)/G_{tot}$), where c represents the cutoff percentile. If the two curves are close together, the model is considered accurate.

Table 4.4: Example of module-order modeling result

c	$G(c)/G_{tot}$	$\hat{G}(c)/G_{tot}$	$\phi(c)$
0.950	0.419	0.361	0.861
0.900	0.631	0.544	0.862
0.850	0.751	0.697	0.928
0.800	0.822	0.772	0.939
0.750	0.884	0.830	0.939
0.700	0.925	0.846	0.915
0.650	0.942	0.871	0.925
0.600	0.963	0.884	0.918
0.550	0.983	0.909	0.924
0.500	1.000	0.913	0.913

- Performance diagram [12] showing the ratio of the predicted ranking and the perfect ranking, $(\phi(c))$ curves. It displays how close the model comes to the perfect ranking. This graph represents the performance of a module-order model. The variation in $\phi(c)$ over a range of c indicates the robustness of the model; small variation implies a robust model.

In this study, the range of cutoff covered at most fifty percent of the modules starting with the most fault-prone. This is because one can hardly imagine applying an enhancement process to more than a half of the modules.

Figure 4.1 presents the curves of a perfect and a predicted ranking of modules according to the results in Table 4.4 and Figure 4.2 shows the ratio of the two lines of Figure 4.1. The cumulative percentage of faults in the modules is plotted over the percentage of modules $(1-c)$ along the horizontal axis, where modules are ordered in

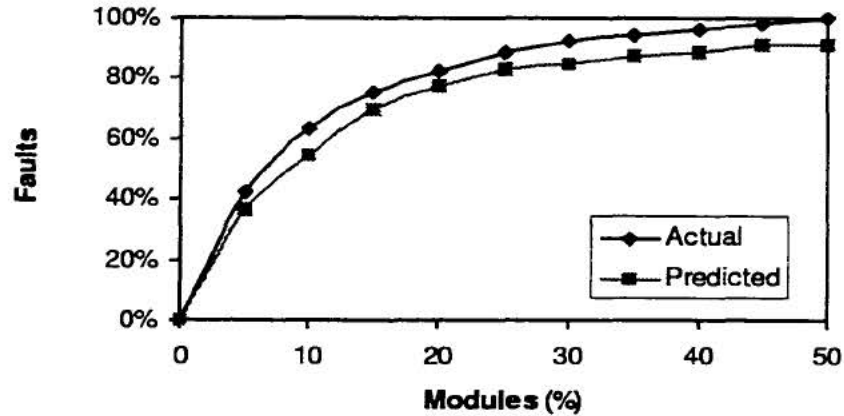


Figure 4.1: Example of Alberg diagram

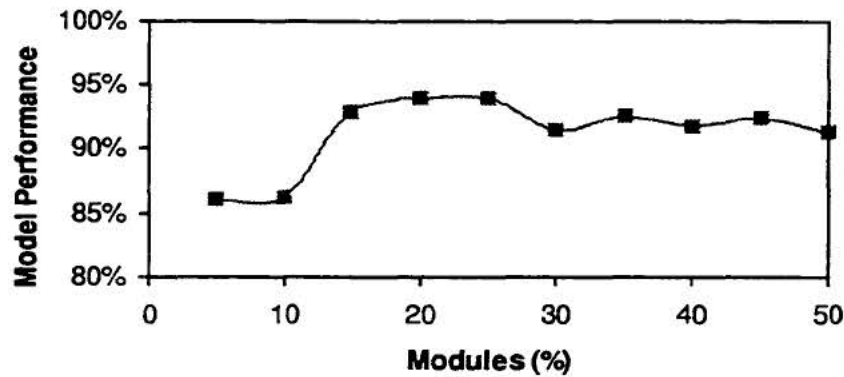


Figure 4.2: Example of Performance diagram

the decreasing order beginning with the most fault-prone. For example, 5% of the modules in the graph represent the cutoff percentile 0.95 in the table.

We order modules beginning with the most fault-prone, the modules with higher ranking will have higher priority for reliability enhancement. Therefore, we will analyze the behavior of a module-order model separated into two ranges.

- Range of higher interest (1 through 25 percentiles): We focused more on this

range because over the half of the faults are contained in this range. Therefore, we used 1 percent increment of modules for observation, and referred to this range as range *I*.

- Range of lower interest (26 through 50 percentiles): modules in this range have lower priority for reliability enhancement than the first one. We used 5 percent increment of modules for observation, and referred to this range as range *II*.

In addition to the two ranges, we gave a closer view on the most critical modules in the highest ranking range (cutoff 1 through 15 percentiles) of range *I*.

We compared the performance of five underlying quantitative models in graphical presentations. However, if the curves from the five techniques were plotted on the same graph, it might be difficult to clearly see the results. Therefore, we showed the graphical presentation by using the following layout.

1. Compare the performance of module-order models based on CBR, MLR, ANN methods, referred as group *I* in one graph.
2. Compare the performance based on tree-modeling methods (CART-LAD, CART-LS and SPLUS), referred as group *II* in another graph.
3. Compare the performance between the two groups. We chose a representative from group *I*, and plot the curve of the selected method compared to the underlying techniques of group *II*.

Table 4.5: Presentation outline for LLTS data

Data set	Test data set	Performance comparison
RAW	Release 2, 3 and 4 (Multiple Releases)	(a) Group <i>I</i> (b) Group <i>II</i> (c) Group <i>I</i> and <i>II</i> (d) Best and worst prediction
PCA	Release 2, 3 and 4 (Multiple Releases)	(a) Group <i>I</i> (b) Group <i>II</i> (c) Group <i>I</i> and <i>II</i> (d) Best and worst prediction

4. Compare the performance based on two underlying models having the best and the worst AAE and ARE value for the two groups (*I* and *II*).

From this layout, we can see the comparative performance of module-order models for both tree-modeling and non-tree-modeling groups. In addition, the hypothesis that better underlying quantitative prediction does not necessarily yield better performance in ordering modules would be obviously proved. This layout were also be used for the remaining data sets, NT and LNTS. Table 4.5 summarizes presentation outline for LLTS data.

4.3.1 Experiments on LLTS-RAW

- **Comparative results of the underlying quantitative models, LLTS-RAW**

In this case, the models were built using LLTS release 1 data set. The dependent variable was the number of faults after unit testing. The best model was

Table 4.6: LLTS-RAW, Comparative accuracy of underlying quantitative models

Model	Release 2		Release 3		Release 4	
	<i>AAE</i>	<i>ARE</i>	<i>AAE</i>	<i>ARE</i>	<i>AAE</i>	<i>ARE</i>
CBR	0.904	0.543	0.917	0.530	0.903	0.533
ANN	0.946	0.584	1.016	0.620	1.249	0.749
MLR	0.890	0.571	0.960	0.602	0.926	0.584
CART-LS	0.948	0.618	0.942	0.602	1.407	0.838
CART-LAD	0.705	0.324	0.803	0.391	0.867	0.419
SPLUS	0.909	0.577	0.954	0.602	1.267	0.774

chosen to be the fitted model based on each underlying techniques. Then, upcoming releases were used to evaluate the model. The results obtained from the five methods are tabulated in Table 4.6 for LLTS-RAW.

Since resubstitution, using the fit as test data set, may yield over optimistic results (quality of fit), release 1 data set was not used to compare the accuracy of the underlying models.

For LLTS-RAW, Mahalonobis distance and Distance weighted average are respectively used as the similarity function and solution algorithm for prediction using CBR. Prior research [29] stated that compared to other similarity functions and solution algorithms, Mahalonobis and distance weighted average provided the best prediction accuracy.

Multiple Linear Regression method used seven independent variables to build the following model [29].

$$\begin{aligned}
\textit{faults} = & 0.0143 \cdot \textit{FILINCUCQ} - 0.0035 \cdot \textit{CNDNOT} + 0.0238 \cdot \textit{NDSENT} \\
& - 0.009 \cdot \textit{NDSEXT} + 0.017 \cdot \textit{NDSPND} + 0.0066 \cdot \textit{NDSINT} \\
& - 0.0031 \cdot \textit{STMDEC}
\end{aligned}$$

The preferred tree model built using the CART-LS method has 10 leaf nodes and utilizes 7 out of 28 independent variables. For the CART-LAD method, the selected tree has 8 leaf nodes and utilizes 7 independent variables. Using SPLUS algorithm, the chosen tree has 23 terminal nodes and utilizes 12 independent variables [27].

From the Table 4.6, the results show that MLR gave better results than CBR and ANN for release 2. CBR provided better prediction accuracy than neural network and multiple linear regression models for release 3 and 4 for both AAE and ARE. For tree modeling methods, CART-LAD results are better than for CART-LS and SPLUS for all test data sets. Further, when comparing all underlying models, CART-LAD had the best prediction among all five techniques. Prior study has shown that CART-LAD is the most effective modeling methodology for prediction and should be preferred among the five underlying techniques for quantitative prediction [27].

- **Comparative results of module-order models, LLTS-RAW**

After retrieving predicted dependent variable from the underlying quantitative models. The results of module-order models are represented in the Alberg and performance diagrams.

1. *LLTS-RAW, comparative results for group I*

The Alberg diagrams shown in Figure 4.3, 4.6 and 4.9 provide an evidence that CBR, MLR and ANN predict the ranking modules significantly close to each other for all three releases. All three curves almost have the same trend even though the prediction accuracy of the underlying quantitative models are quite different.

Figure 4.4, 4.7 and 4.10 give us a close view of the module-order models' behavior for the most critical modules regarding software reliability. CBR and ANN give slightly closer ranking to the perfect ranking than MLR for the cutoff 1-5 percentile for all releases. Consequently, MLR does not perform as well as the two other methods for the beginning of the cutoff range illustrated in Figure 4.5, 4.8 and Figure 4.11. Further for release 4, ANN does not have the best predicted ranking for the 3-10 percentile cutoff. Therefore, we see a gap between ANN and the two other methods in the performance diagram at those particular percentiles.

When analyzing the performance on range *I*, all three techniques perform very close to each other over the range. The three methods only present different performance at the beginning of the cutoff percentile. Precisely, both CBR and ANN perform better than MLR at the beginning of the cutoff range. When considering range *II*, the three techniques also present close performance over the range for all releases.

Since group *I*'s techniques perform very close when module-order modeling, we can not determine which technique presents the best performance in group *I*. This depends on the particular cutoff percentile the manager will choose. For example, MLR performs better than CBR and ANN for release 3 for the cutoff 10-25 percentile. However, both CBR and ANN perform better than MLR for the cutoff 1-5 percentile for the same release.

In addition, CBR provided better prediction than MLR and ANN for release 3, but did not perform as well as MLR and ANN for the main part of the considered ranges. This confirms our hypothesis that better prediction doesn't always yield better performance when module-order modeling. Comparative performance of group *I* is illustrated in Figure 4.5, 4.8 and 4.11.

2. *LLTS-RAW, Comparative Results for Group II*

In contrast to group *I*, the three techniques in group *II* present the different predicted rankings as shown in Figure 4.12, 4.15 and 4.18.

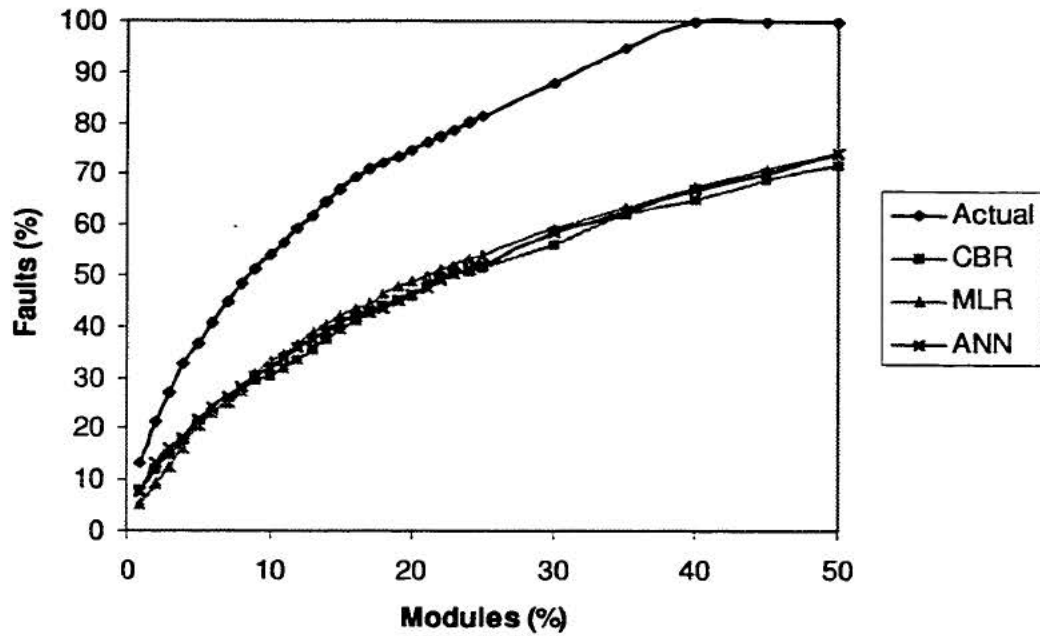


Figure 4.3: Alberg diagram for LLTS-RAW release 2: CBR, MLR, ANN

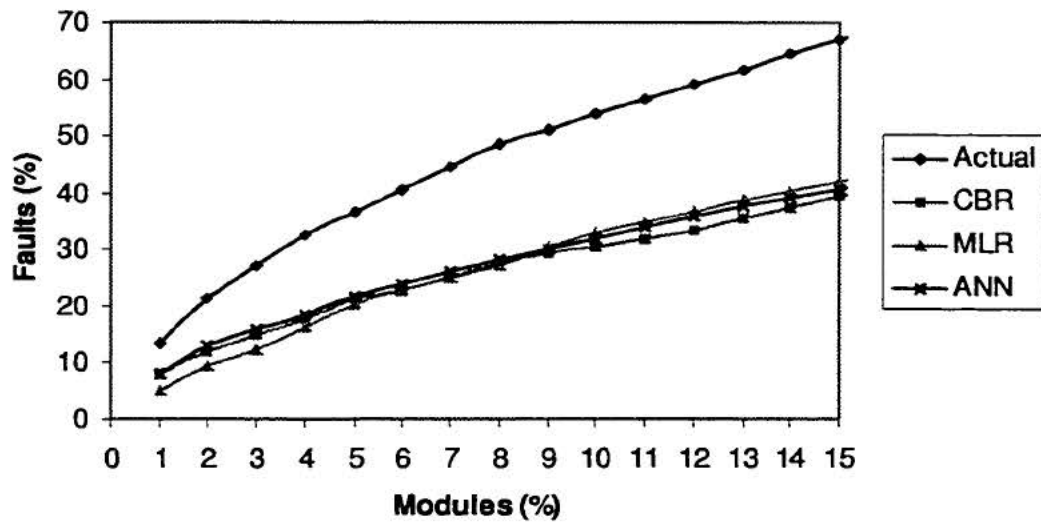


Figure 4.4: Close view of Alberg diagram for LLTS-RAW release 2: CBR, MLR, ANN

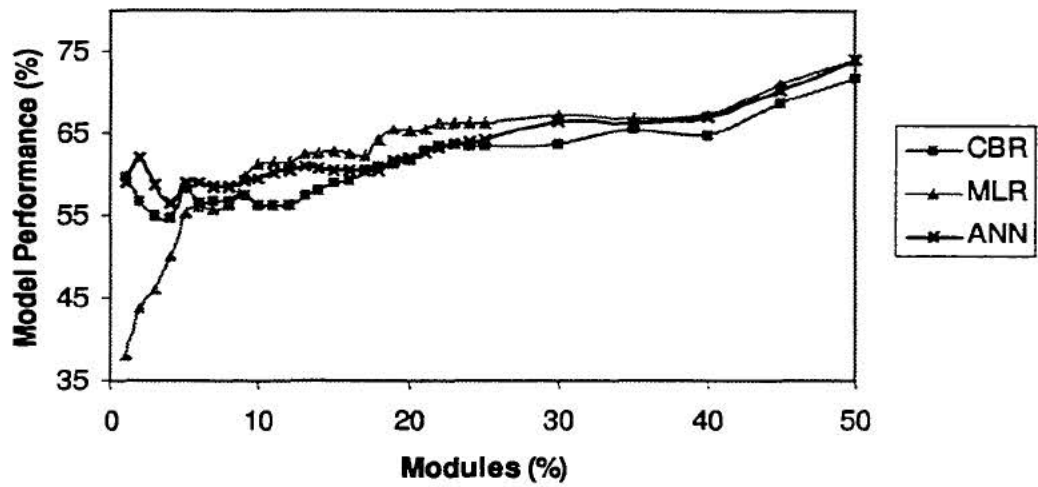


Figure 4.5: Performance of LLTS-RAW release 2: CBR, MLR, ANN

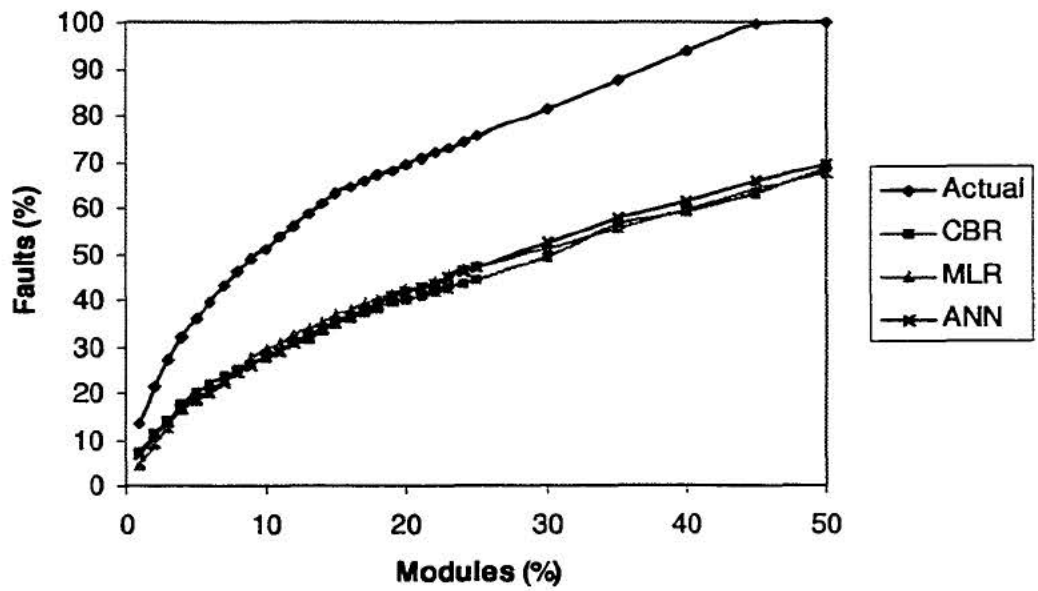


Figure 4.6: Alberg diagram for LLTS-RAW release 3: CBR, MLR, ANN

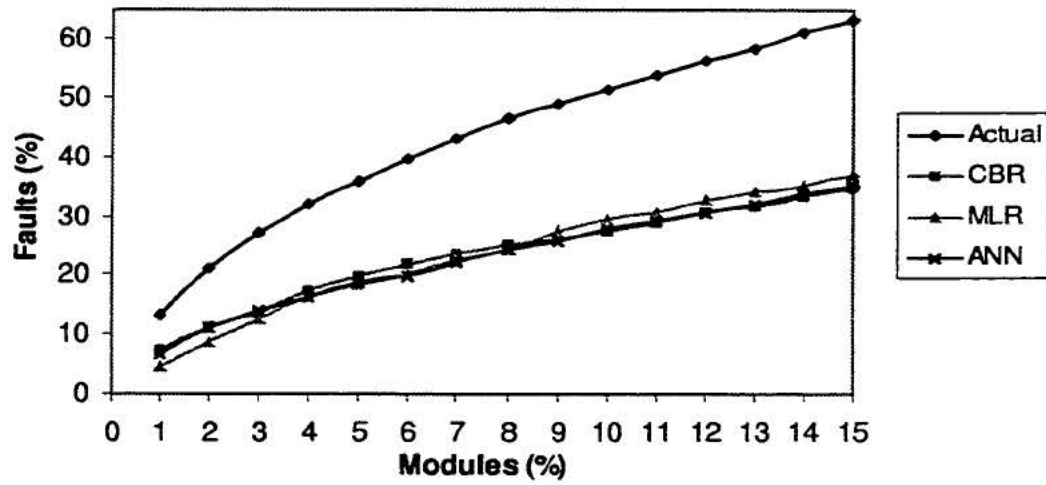


Figure 4.7: Close view of Alberg diagram for LLTS-RAW release 3: CBR, MLR, ANN

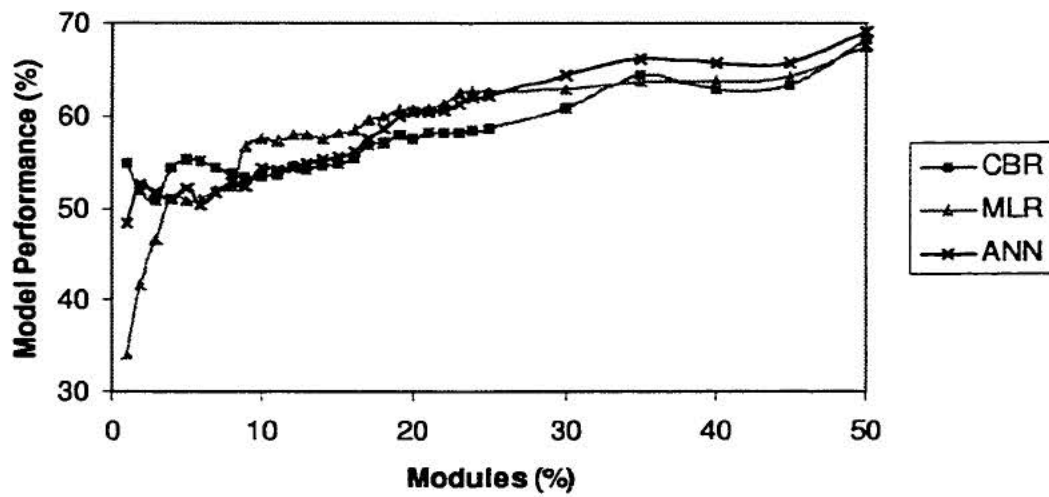


Figure 4.8: Performance of LLTS-RAW release 3: CBR, MLR, ANN

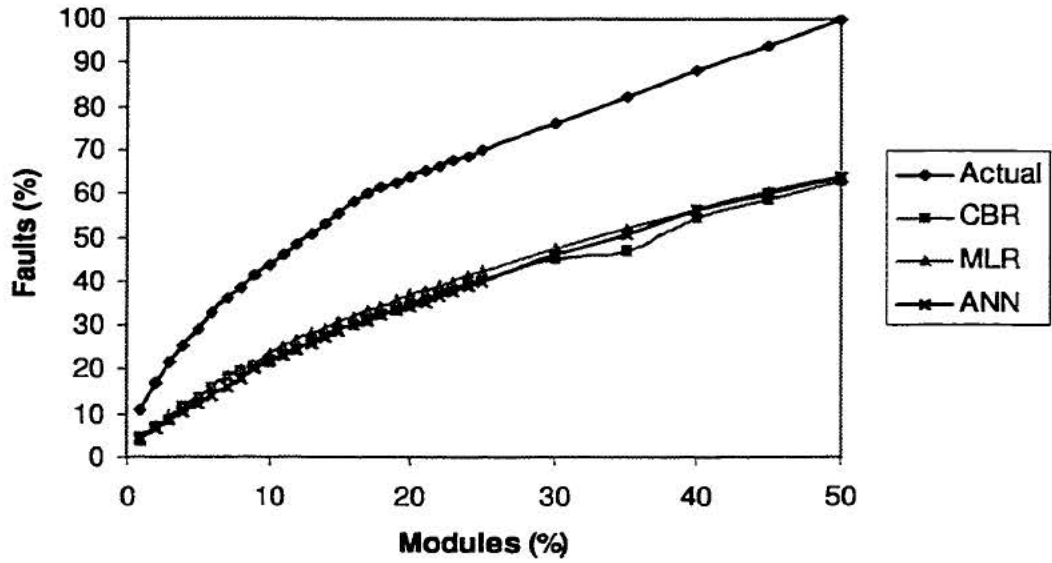


Figure 4.9: Alberg diagram for LLTS-RAW release 4: CBR, MLR, ANN

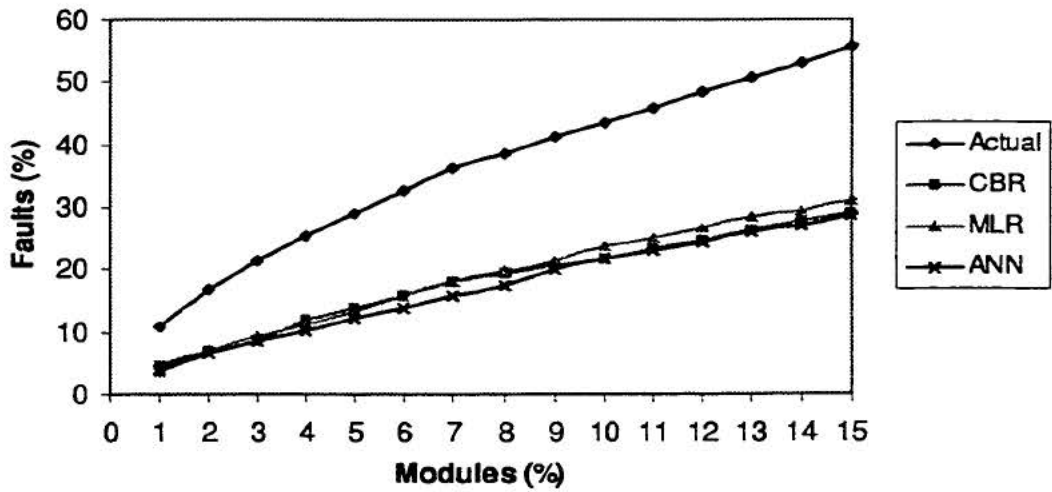


Figure 4.10: Close view of Alberg diagram for LLTS-RAW release 4: CBR, MLR, ANN

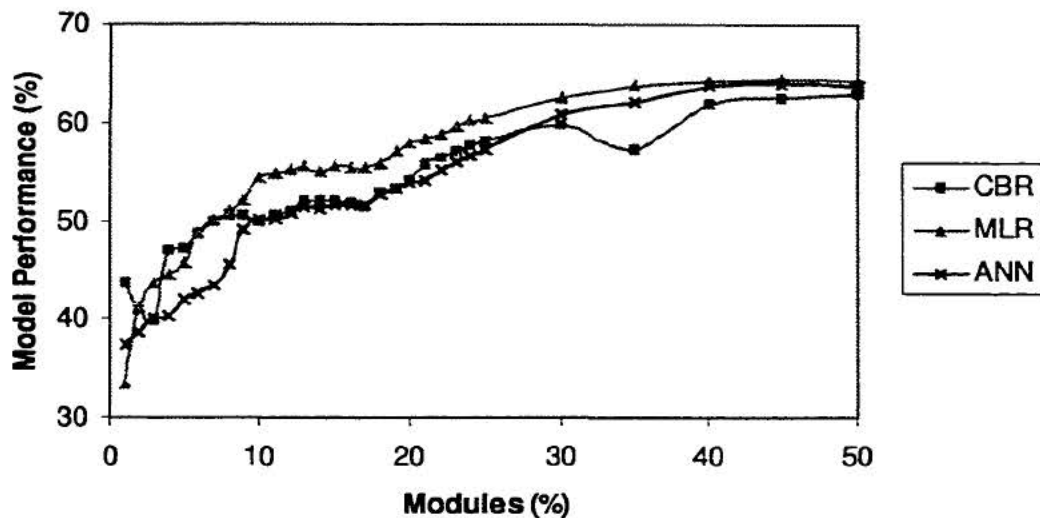


Figure 4.11: Performance of LLTS-RAW release 4: CBR, MLR, ANN

The close views in Figure 4.13, 4.16 and 4.19, show that all group *II*'s techniques give close predicted rankings for the first half of the critical range (1-7 percentile). However, CART-LAD does not give a ranking closer to the perfect ranking than the other two methods for the second half. We see that the performance of CART-LAD declines compared to SPLUS and CART-LS after that first half of the critical range for all releases.

When considering performances for range *I*, CART-LS and SPLUS perform close to each other. However, CART-LAD performs obviously not as well as the former two methods for all releases. This is noticeable from the big gaps in Figure 4.14, 4.17 and 4.20.

For range *II*, all techniques perform completely different from each other.

The two CART methods present high variation of performance for this range. CART-LS presents the best performance at the end of range *II*. The performance of CART-LAD is much better compared to its performance on range *I*. As a consequence, CART-LAD performs better than SPLUS at the end of range *II*. For SPLUS, the technique performs consistently for both ranges *I* and *II*. This infers that SPLUS has the least variation of $\phi(c)$ and generates the most robust model in group *II*.

Furthermore, CART-LAD had the best prediction accuracy for all releases, but it does not present the best module-order modeling results compared to the other two methods as described above for the range *I*. This case obviously confirms our hypothesis.

3. *LLTS-RAW, Group I and Group II Models Comparison*

Since the three methods in group *I* perform close to each other when module-order modeling, we chose one of them to compare with the tree-modeling group. Since CBR performs consistently over the considered ranges, we chose it to represent group *I*'s techniques.

We can obviously observe that CBR has a module-ordering behavior close to SPLUS in all releases as displayed in Figure 4.21, 4.24 and 4.27.

For the close view of the most critical range, CBR gives the nearest ranking to the perfect ranking at the starting cutoff range (1-3 percentile) illustrated

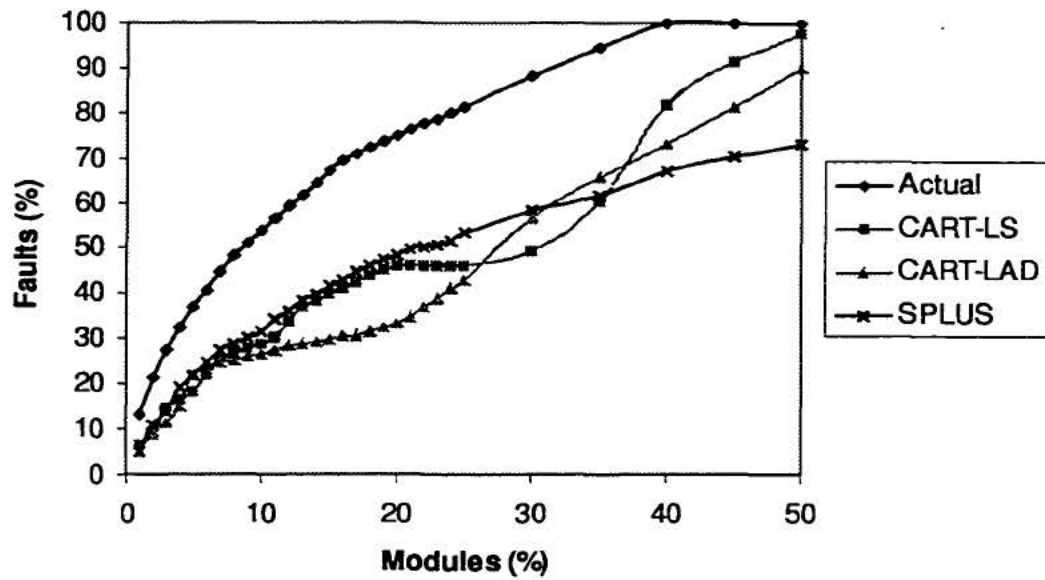


Figure 4.12: Alberg diagram for LLTS-RAW release 2: CART-LS, CART-LAD, SPLUS

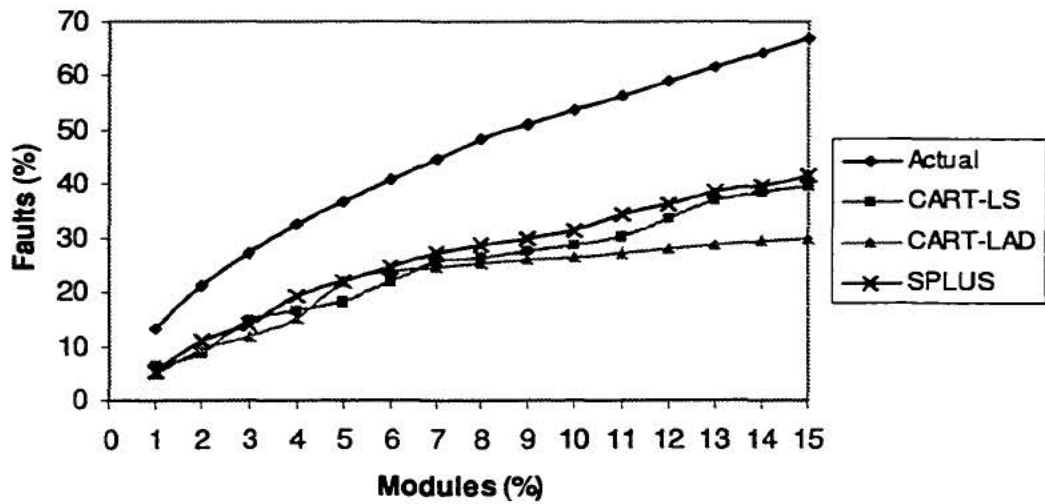


Figure 4.13: Close view of Alberg diagram for LLTS-RAW release 2: CART-LS, CART-LAD, SPLUS

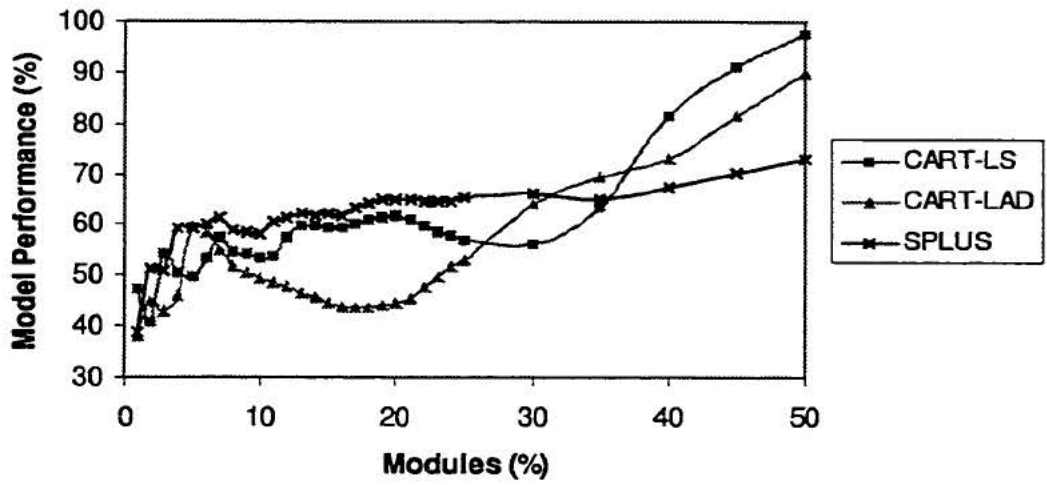


Figure 4.14: Performance of LLTS-RAW release 2: CART-LS, CART-LAD, SPLUS

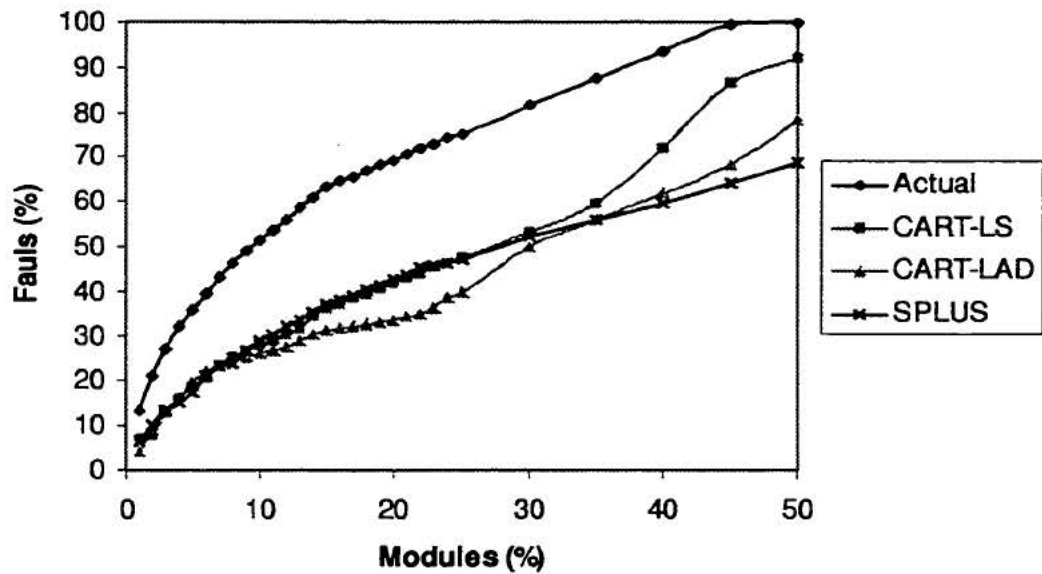


Figure 4.15: Alberg diagram for LLTS-RAW release 3: CART-LS, CART-LAD, SPLUS

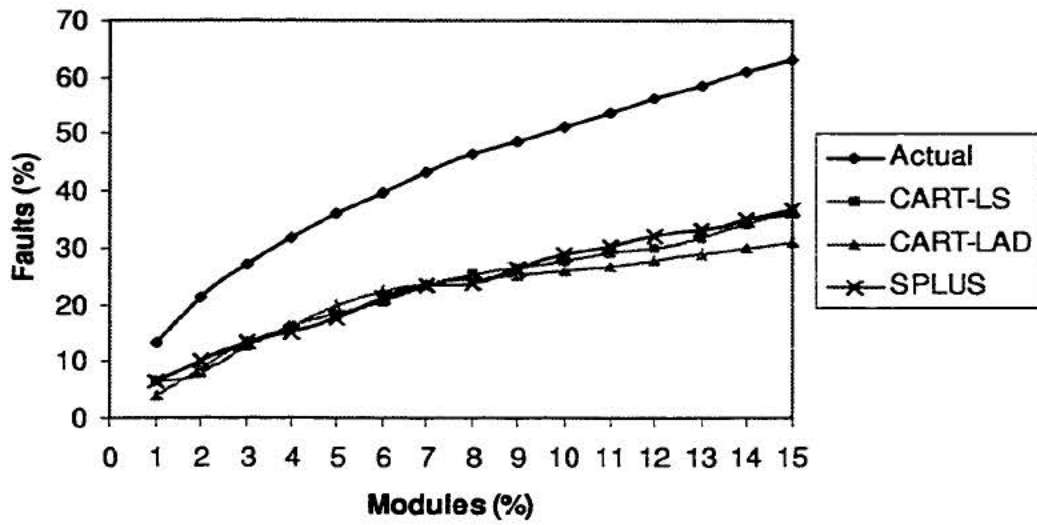


Figure 4.16: Close view of Alberg diagram for LLTS-RAW release 3: CART-LS, CART-LAD, SPLUS

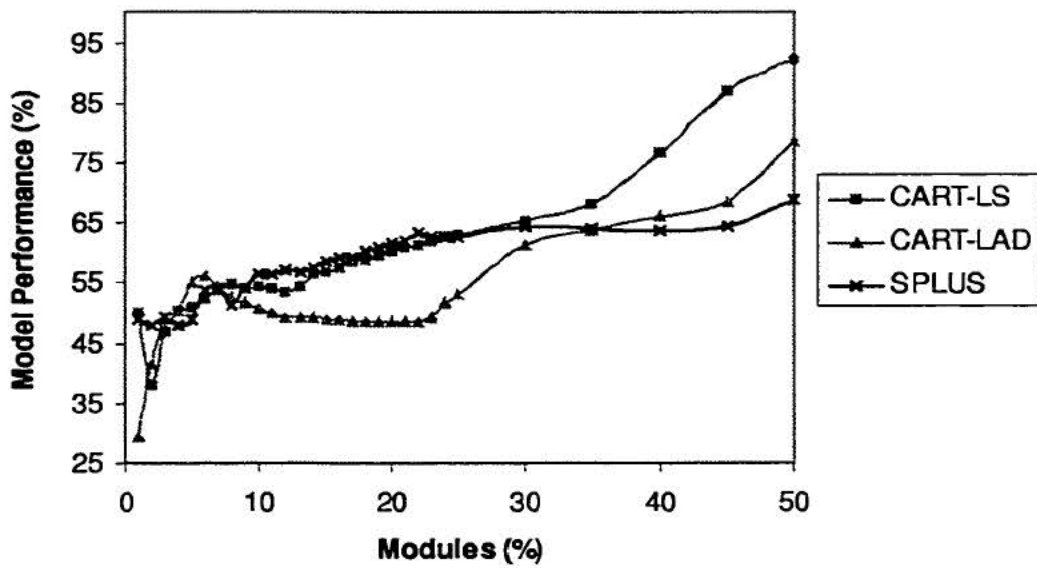


Figure 4.17: Performance of LLTS-RAW release 3: CART-LS, CART-LAD, SPLUS

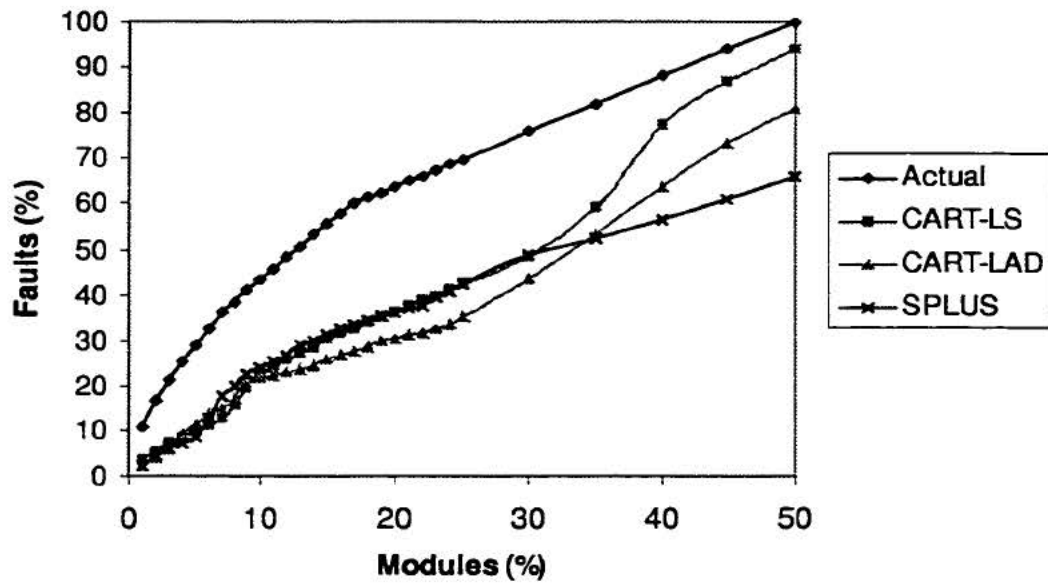


Figure 4.18: Alberg diagram for LLTS-RAW release 4: CART-LS, CART-LAD, SPLUS

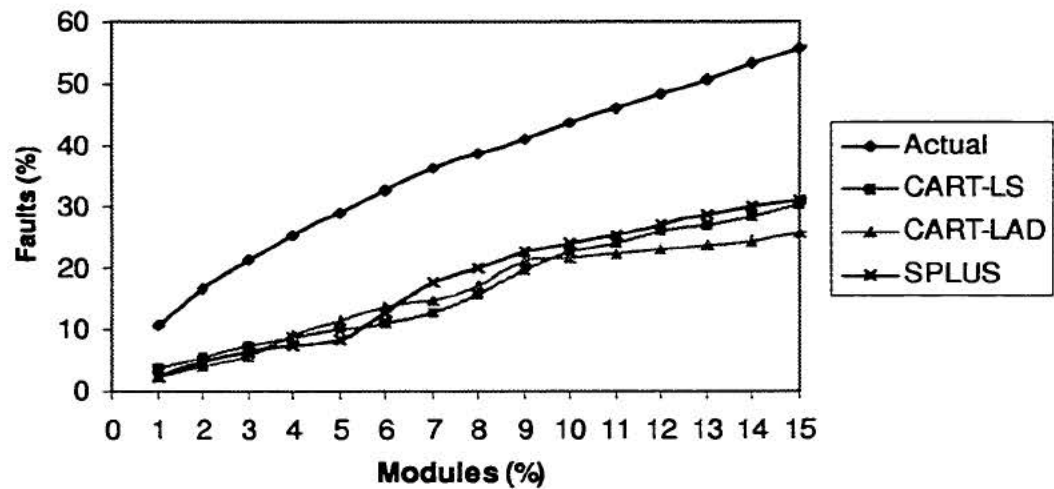


Figure 4.19: Close view of Alberg diagram for LLTS-RAW release 4: CART-LS, CART-LAD, SPLUS

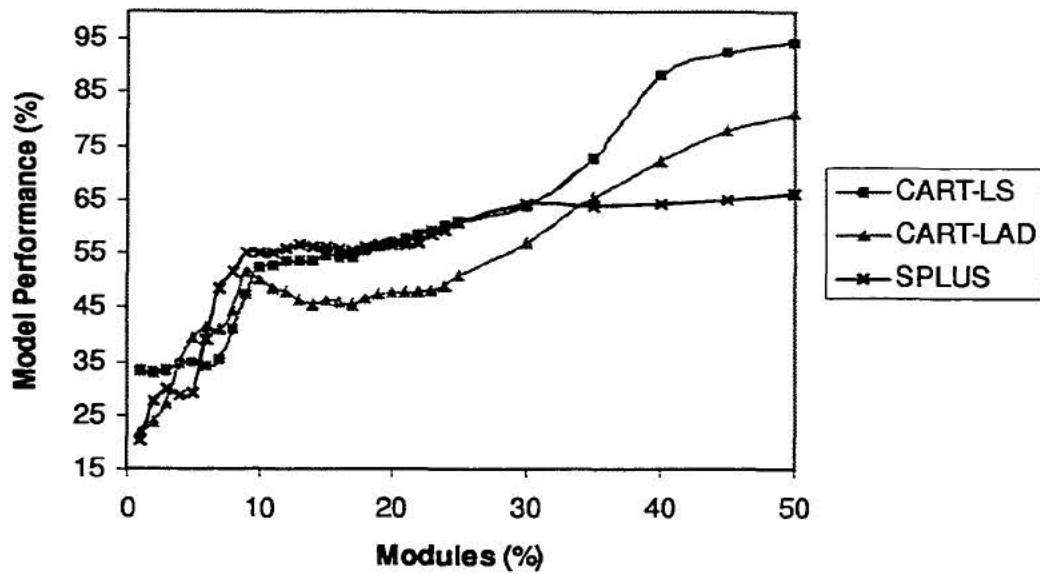


Figure 4.20: Performance of LLTS-RAW release 4: CART-LS, CART-LAD, SPLUS

in Figure 4.22, 4.25 and 4.28, so CBR performs better than all group *I*'s methods for the beginning cutoff range for all releases.

Figure 4.23, 4.26 and 4.29 show the comparative performance between CBR and group *II*'s techniques. The diagrams show that SPLUS performs close to CBR for all releases compared to the two CART methods. This infers that SPLUS also performs close to all techniques in group *I* when module-order modeling for LLTS-RAW data.

4. LLTS-RAW, comparative results regarding AAE and ARE

CART-LAD gave the most accurate prediction among the five modeling techniques indicated by AAE and ARE values in Table 4.6. The results show that

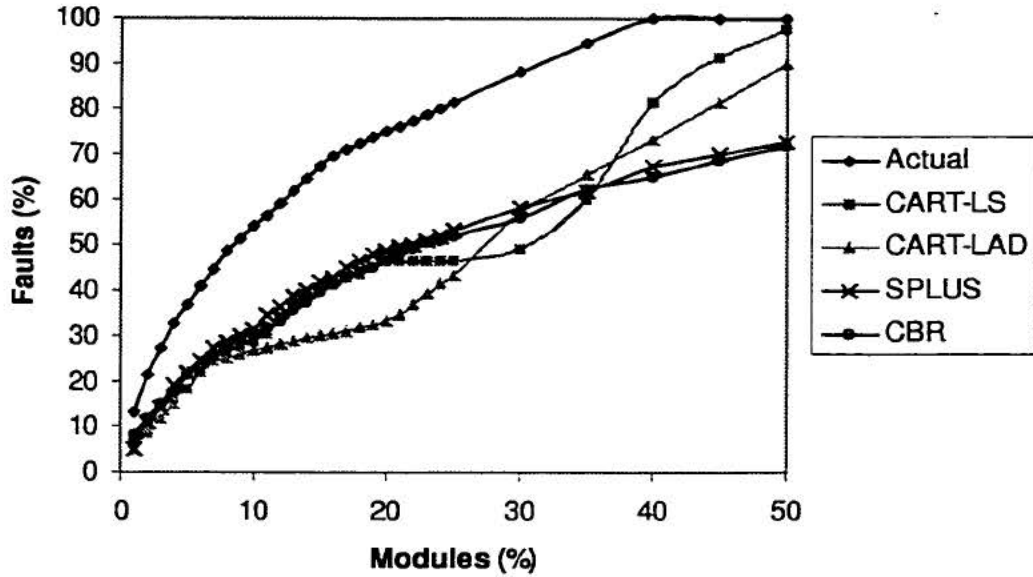


Figure 4.21: Alberg diagram of LLTS-RAW release 2: CART-LS, CART-LAD, SPLUS and CBR

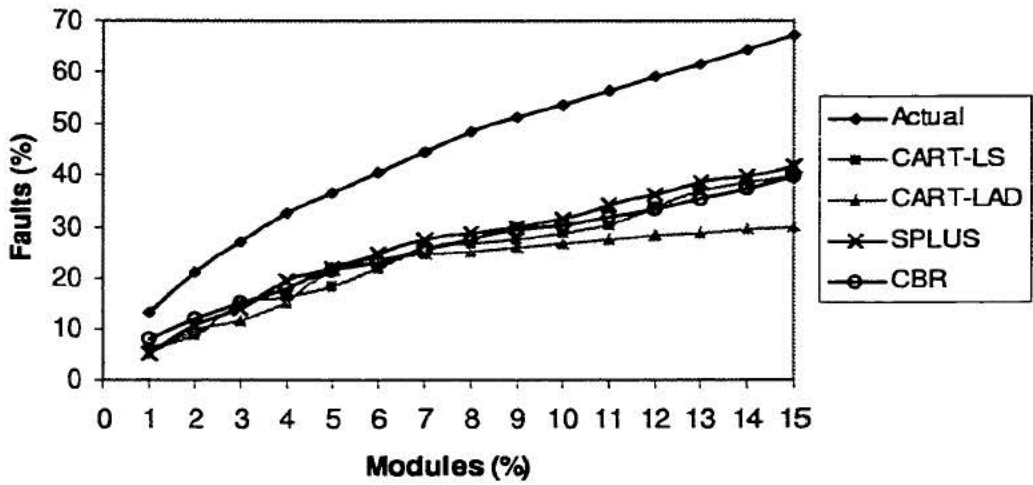


Figure 4.22: Close view of Alberg diagram of LLTS-RAW release 2: CART-LS, CART-LAD, SPLUS and CBR

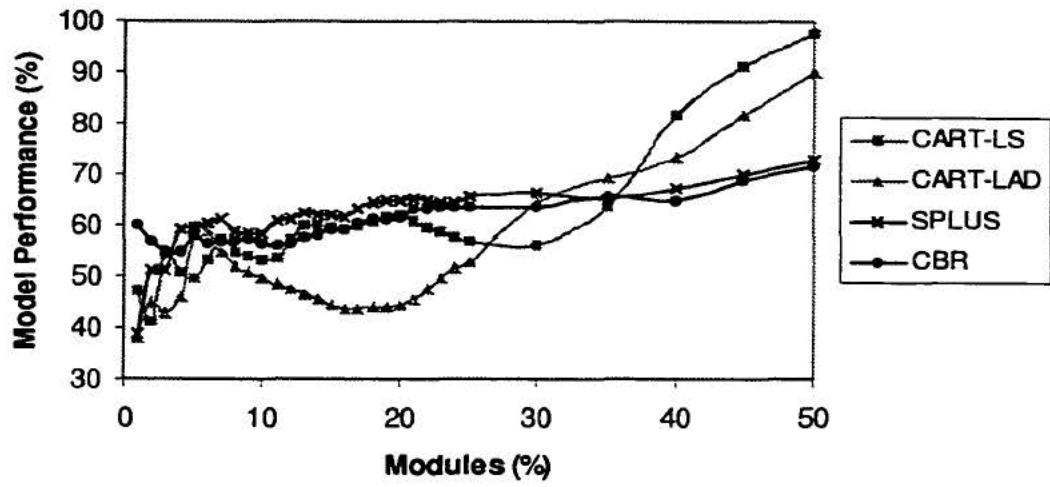


Figure 4.23: Performance of LLTS-RAW release 2: CART-LS, CART-LAD, SPLUS and CBR

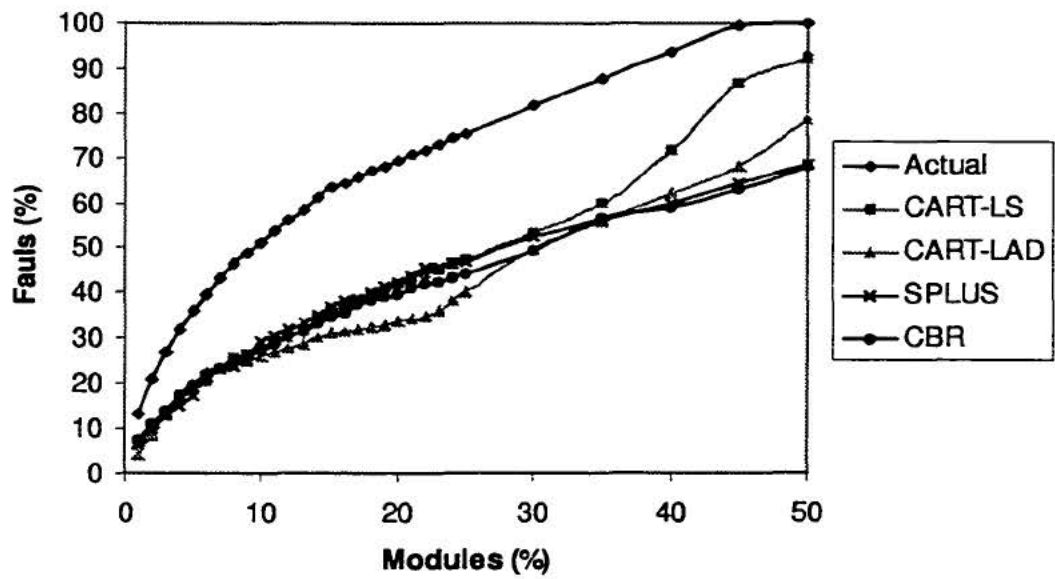


Figure 4.24: Alberg diagram for LLTS-RAW release 3: CART-LS, CART-LAD, SPLUS and CBR

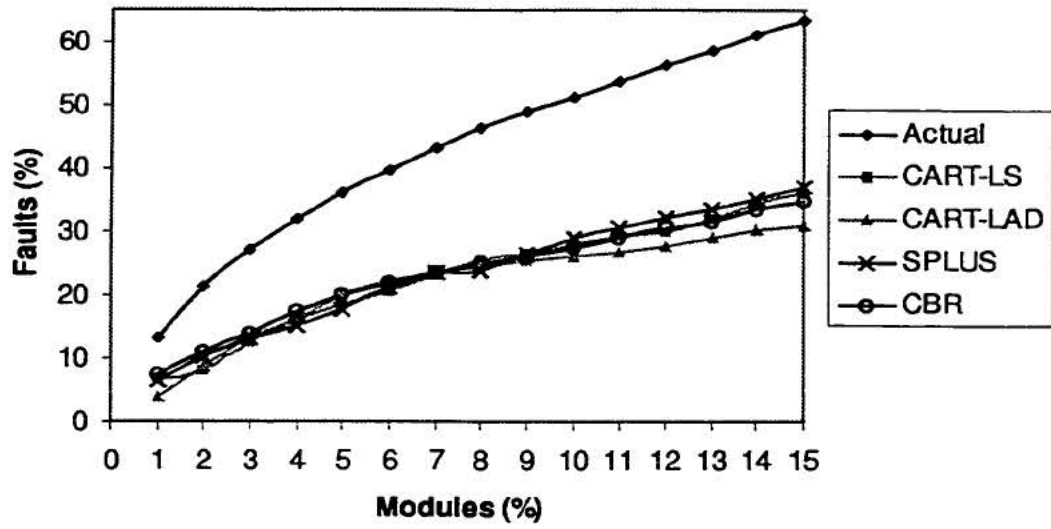


Figure 4.25: Close view of Alberg diagram for LLTS-RAW release 3: CART-LS, CART-LAD, SPLUS and CBR

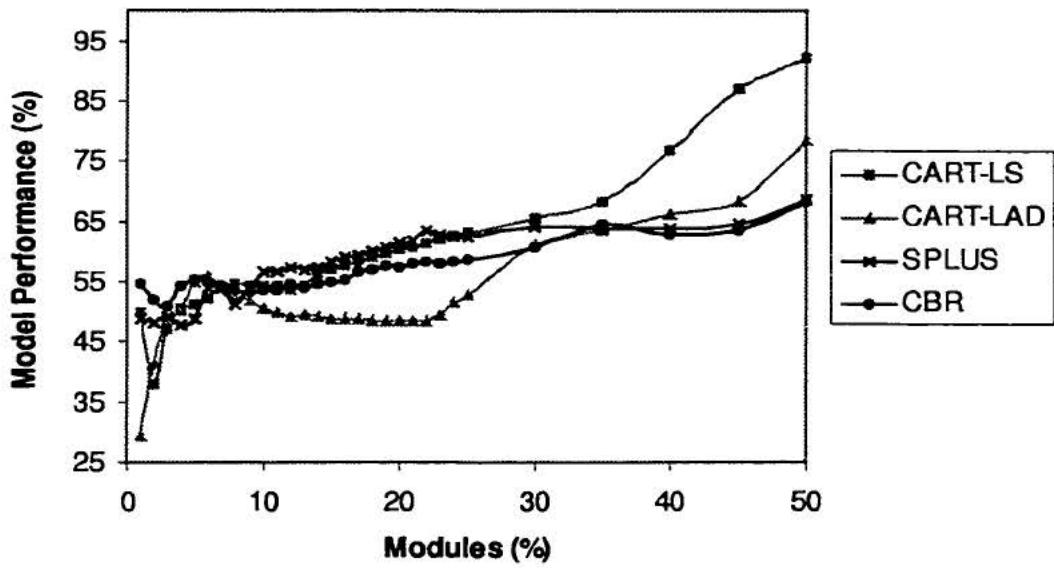


Figure 4.26: Performance of LLTS-RAW release 3: CART-LS, CART-LAD, SPLUS and CBR

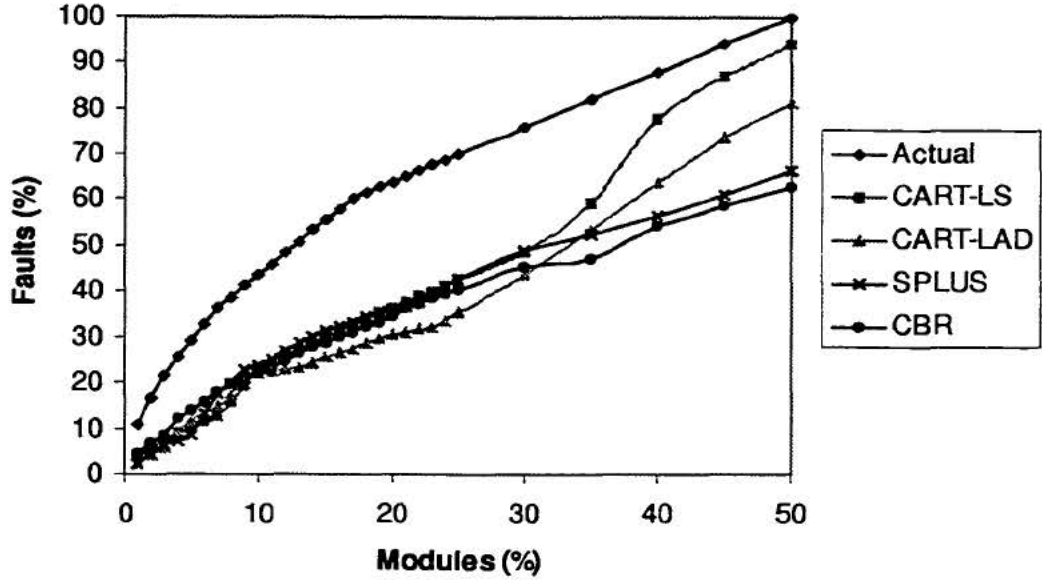


Figure 4.27: Alberg diagram for LLTS-RAW release 4: CART-LS, CART-LAD, SPLUS and CBR.

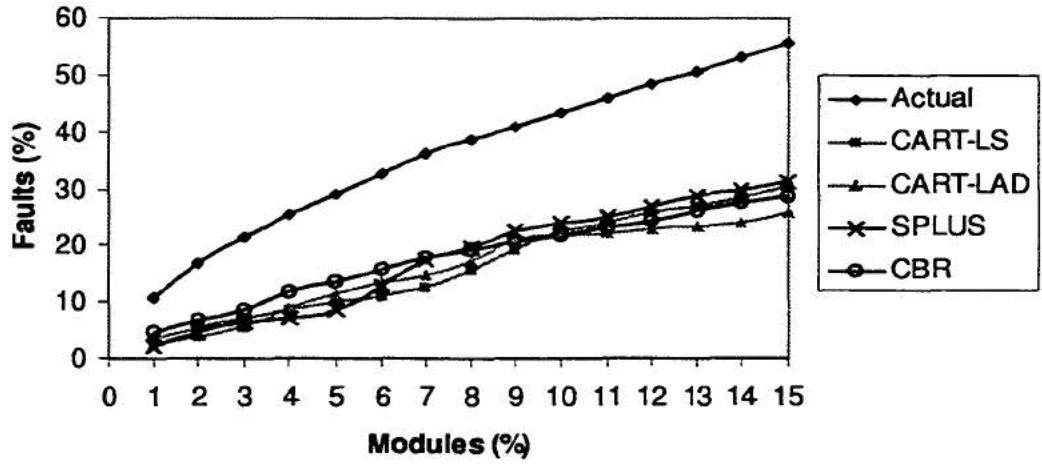


Figure 4.28: Close view of Alberg diagram for LLTS-RAW release 4: CART-LS, CART-LAD, SPLUS and CBR.

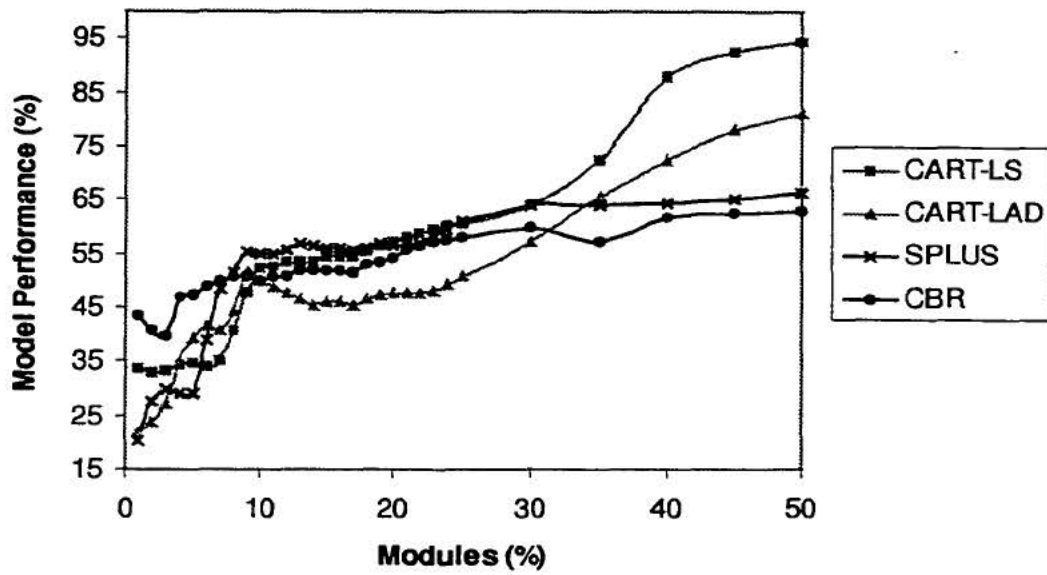


Figure 4.29: Performance of LLTS-RAW release 4: CART-LS, CART-LAD, SPLUS and CBR

it had the best prediction accuracy for all releases of the LLTS-RAW data set. We compare CART-LAD to the models that bring the worst prediction accuracy for the three releases. For release 2 and 4, CART-LS had the worst accuracy, and SPLUS had the worst accuracy for release 3.

The Alberg diagrams show that CART-LAD does not present the predicted rankings closer to the perfect ranking than CART-LS and SPLUS. For release 2, CART-LAD and CART-LS alternatively present closer ranking to the perfect ranking over both ranges *I* and *II* as illustrated in Figure 4.30. For release 3 and 4, the predicted rankings based on ANN and SPLUS are closer to the perfect ranking than those based on CART-LS over ranges *I* and *II* as

illustrated in Figure 4.33 and 4.36.

When focusing on the most critical range, the selected two techniques alternatively predict closer rankings to the perfect rankings as illustrated in Figure 4.31, 4.34 and 4.37. Therefore, we can observe that neither method clearly outperforms the other for the most critical modules.

When considering the performance, CART-LAD does not perform better than the other techniques with poorer prediction accuracy when module-order modeling. For release 2, CART-LS performs considerably better than CART-LAD over range *I*. For release 3, ANN also performs better than CART-LAD over range *I*. For release 4, CART-LS almost has better performance than CART-LAD over both ranges *I* and *II*. The performance diagrams are shown in Figure 4.32, 4.35 and 4.38. These cases provide more evidences that better underlying quantitative prediction doesn't always yield better performance when module-order modeling.

4.3.2 Experiment on LLTS-PCA

- **Comparative results of the underlying quantitative models, LLTS-PCA**

Using the same strategy as for LLTS-RAW, release 1 was used as the fit data set and subsequent releases were used as the test data sets. Application of the models

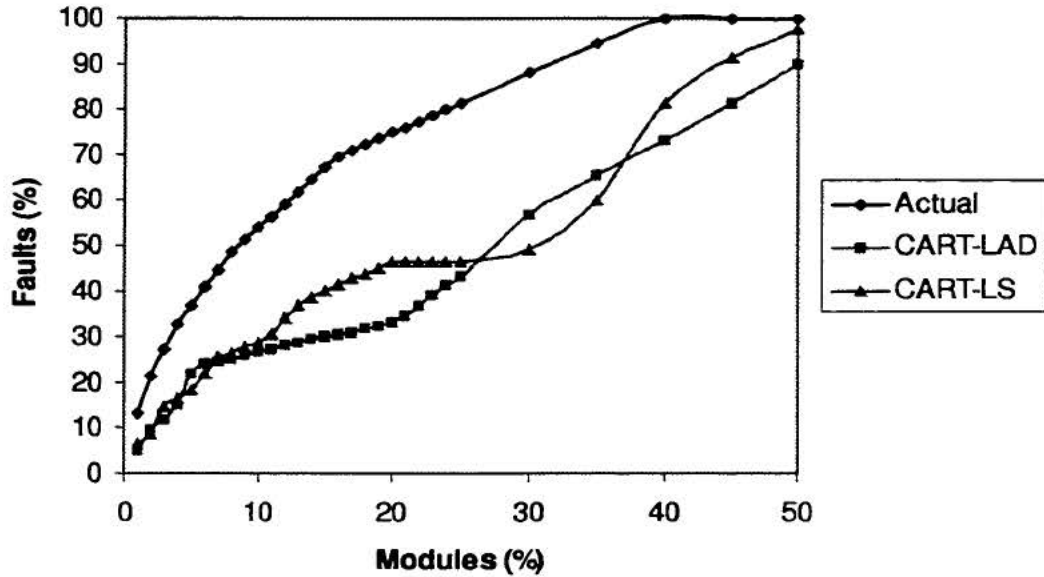


Figure 4.30: Alberg diagram of LLTS-RAW release 2: CART-LS, CART-LAD

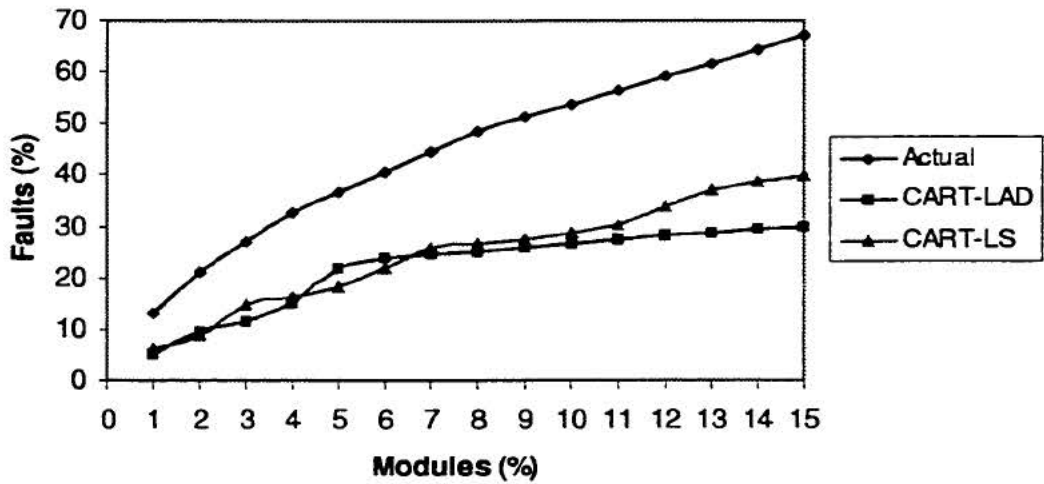


Figure 4.31: Close view of Alberg diagram for LLTS-RAW release 2: SPLUS, CART-LAD

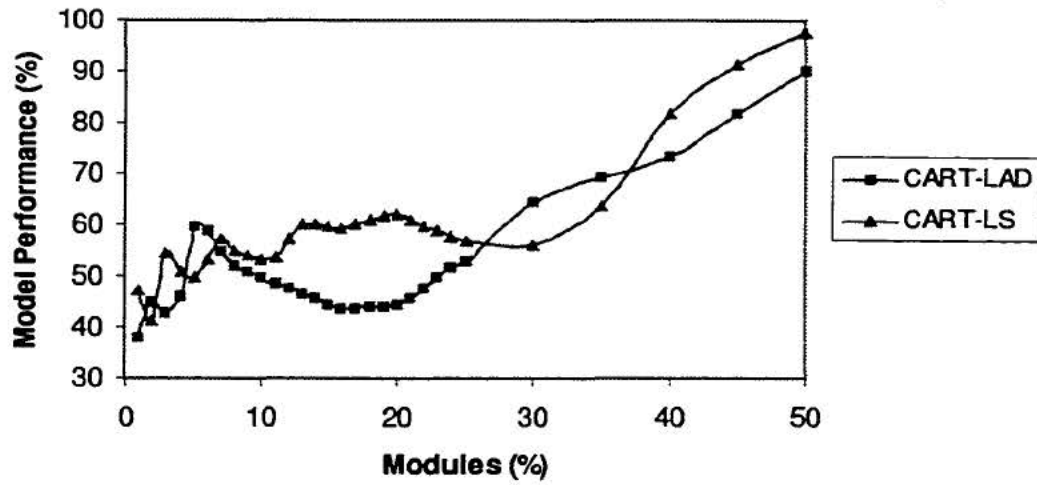


Figure 4.32: Performance of LLTS-RAW release 2: CART-LS, CART-LAD

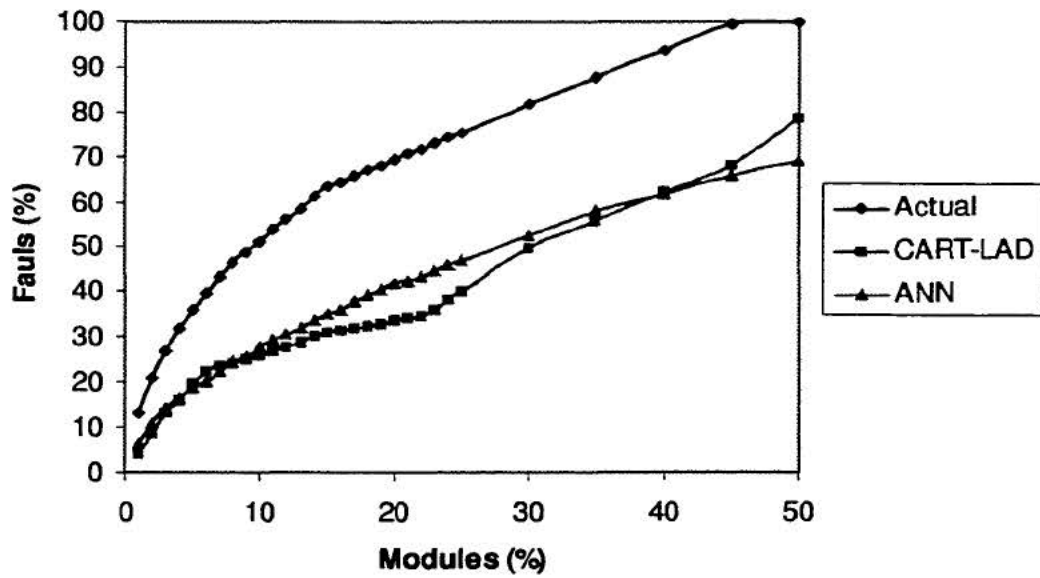


Figure 4.33: Alberg diagram for LLTS-RAW release 3: SPLUS, CART-LAD

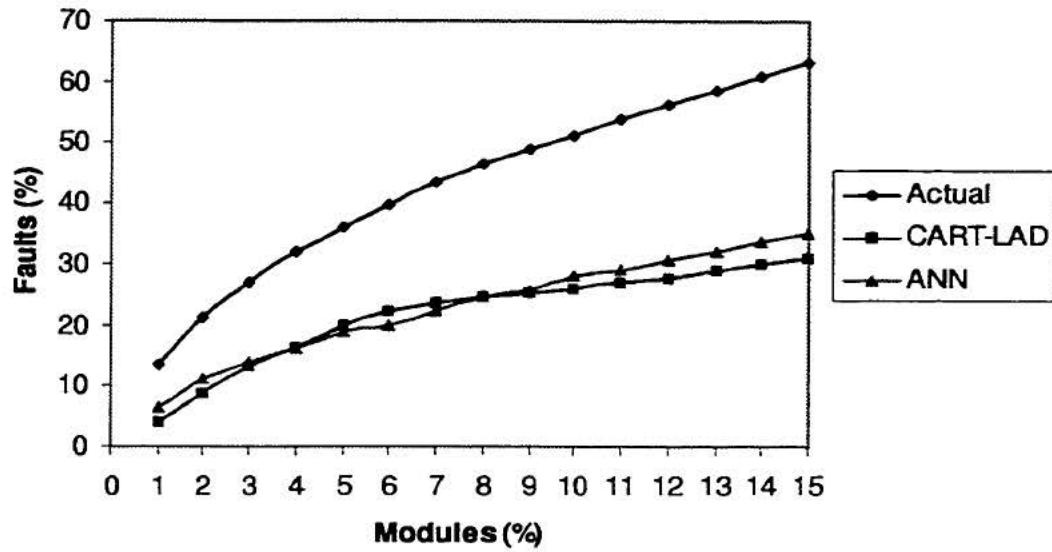


Figure 4.34: Close view of Alberg diagram for LLTS-RAW release 3: SPLUS, CART-LAD

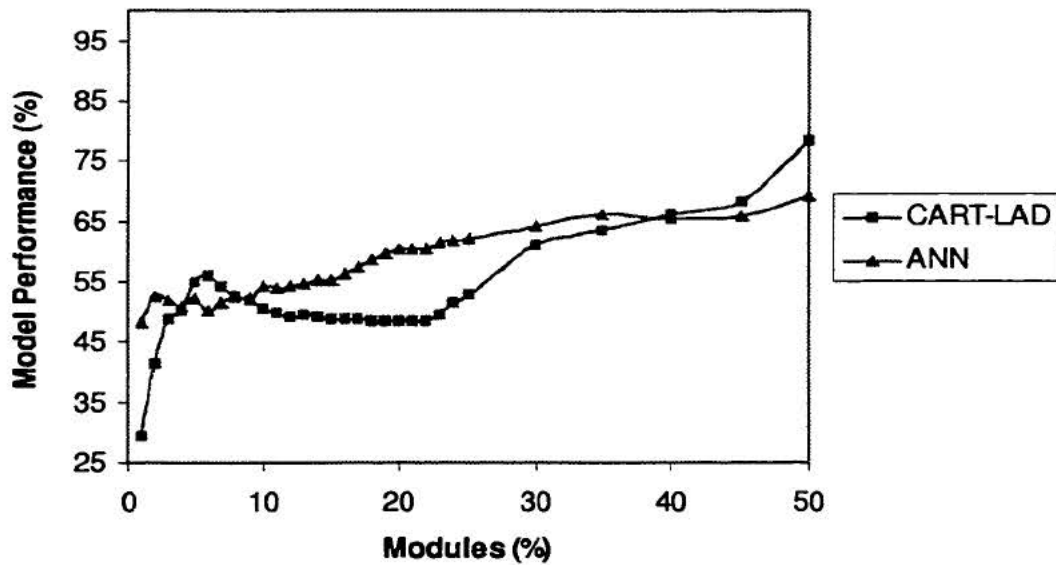


Figure 4.35: Performance of LLTS-RAW release 3: SPLUS, CART-LAD

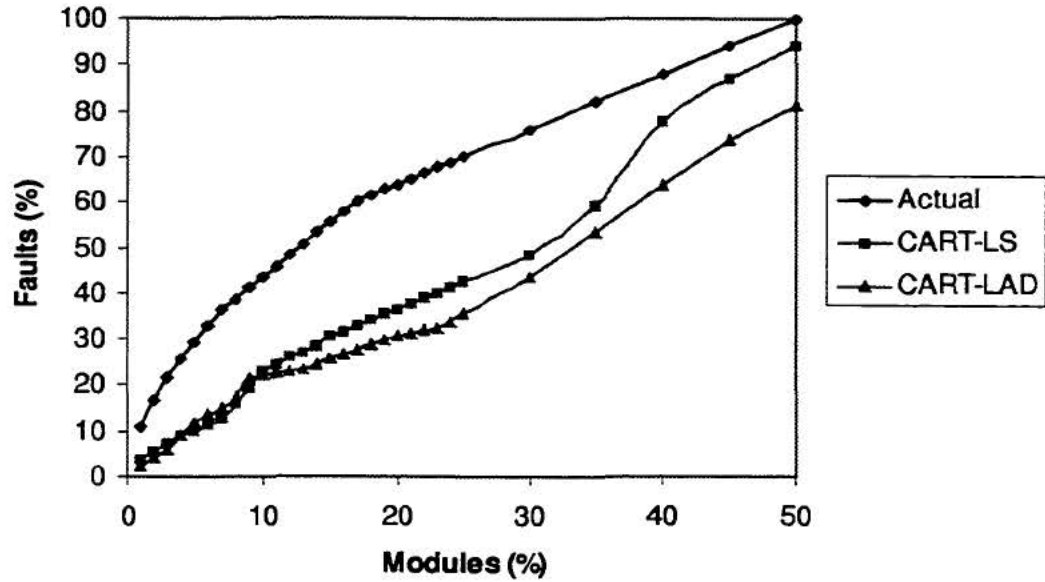


Figure 4.36: Alberg diagram for LLTS-RAW release 4: CART-LS, CART-LAD

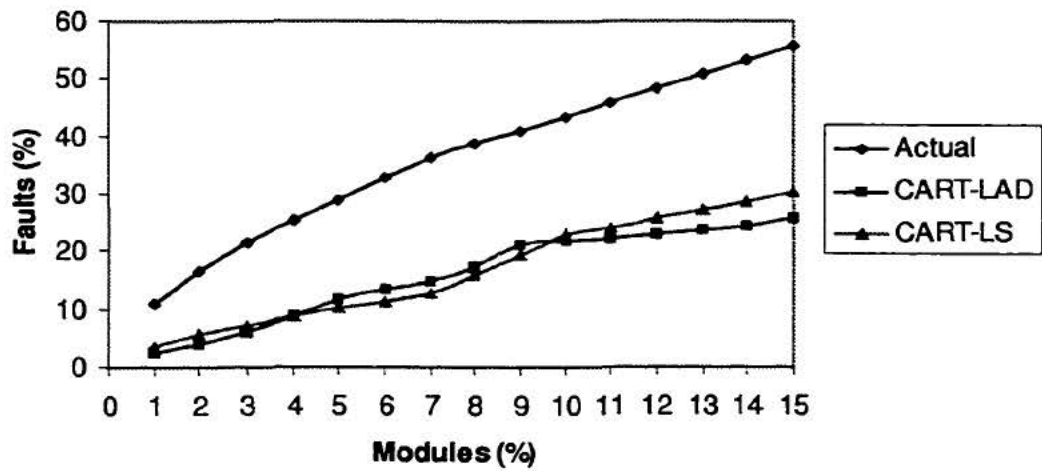


Figure 4.37: Close view of Alberg diagram for LLTS-RAW release 4: SPLUS, CART-LAD

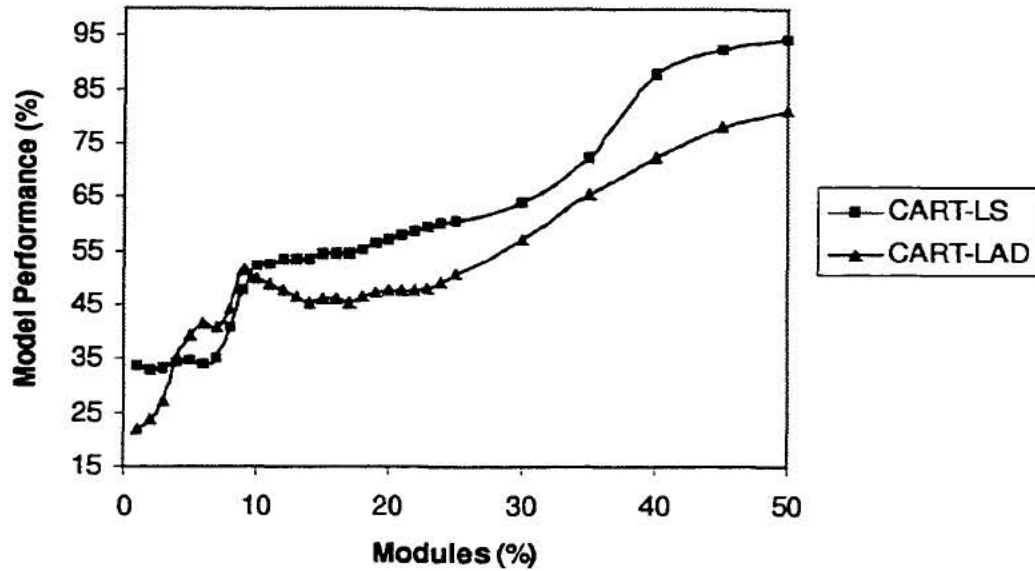


Figure 4.38: Performance of LLTS-RAW release 4: CART-LS, CART-LAD

Table 4.7: LLTS-PCA, Comparative accuracy of underlying quantitative models

Model	Release 2		Release 3		Release 4	
	<i>AAE</i>	<i>ARE</i>	<i>AAE</i>	<i>ARE</i>	<i>AAE</i>	<i>ARE</i>
CBR	0.835	0.523	0.871	0.519	0.810	0.477
ANN	0.887	0.555	0.948	0.576	0.989	0.615
MLR	0.875	0.567	0.976	0.626	0.954	0.637
CART-LS	0.972	0.647	0.975	0.633	1.113	0.682
CART-LAD	0.727	0.344	0.823	0.408	0.860	0.456
SPLUS	0.925	0.602	0.973	0.621	1.568	0.949

based on different underlying methodologies to the test data set of LLTS-PCA is shown in Table 4.7.

For LLTS-PCA, the city-block distance with distance weighted solution algorithm gave the most accurate prediction for CBR [29]. MLR results used six

independent variables with intercept to build the following model [29].

$$\begin{aligned} faults = & 0.7915 + 0.3745 \cdot USAGE + 0.847 \cdot PROD1 + 0.3985 \cdot PROD2 \\ & + 0.2135 \cdot PROD3 + 0.3082 \cdot PROD4 + 0.1236 \cdot PROD5 \\ & + 0.1494 \cdot PROD6 \end{aligned}$$

The chosen tree using CART-LS has 7 terminal nodes and uses 3 out of 10 independent variables. The preferred tree using CART-LAD has 8 terminal nodes and utilizes 4 out of 10 independent variables. For SPLUS, the selected tree has 26 terminal nodes and uses 9 out of 10 independent variables to build the tree [27].

According to Table 4.7, CBR has the best prediction accuracy compared to MLR and ANN for all three releases in group *I*. For group *II*, CART-LAD gave significantly better results than CART-LS and SPLUS. Further, when comparing all underlying models, CART-LAD presented the best prediction for release 2 and 3 while CBR had the best prediction for release 4.

- **Comparative results of module-order models, LLTS-PCA**

1. *LLTS-PCA, Comparative Results for Group I*

Figure 4.39 and 4.42 denote that CBR, MLR and ANN generate very close rankings for all releases. They almost present identical predicted ranking over ranges *I* and *II* for release 2 and 3. For release 4, the three techniques present close ordering for range *I*, but MLR and ANN are closer to the perfect ranking than CBR for range *II* depicted in Figure 4.45.

Figure 4.40, 4.43 and 4.46 give us a closer view to the most critical modules. CBR and ANN give a closer ranking to the perfect ranking than MLR at the beginning of the critical range (1-3 percentile) for all releases. Hence, the performance of MLR is not as good as the two other methods at that starting range. ANN, in contrast to MLR, gives the nearest module-order model to the perfect ranking at that specific range for release 2 and 3. However, ANN gives the farthest ranking from the perfect ranking at the cutoff 3-7 percentile for release 4. The module-order models based on the three group *I*'s methods are very close to each other after the cutoff 7 percentile.

When analyzing the performances for the first two releases, we notice that the models of group *I* perform close to each other over both ranges *I* and *II*. The three methods present different performances only for the beginning of the cutoff range, 1-5 percentile, illustrated in Figure 4.41 and 4.44.

When considering release 4, MLR and ANN perform better than CBR for the second half of range *I* and all over range *II* even though CBR provided the most accurate prediction in group *I* for this release, shown in Figure 4.47. This case further validates the hypothesis that an underlying model with better prediction accuracy, does not necessarily indicate a better module-order model.

To summarize, all techniques in group *I* perform close to each other. They present parallel trends of performance in both the Alberg and performance

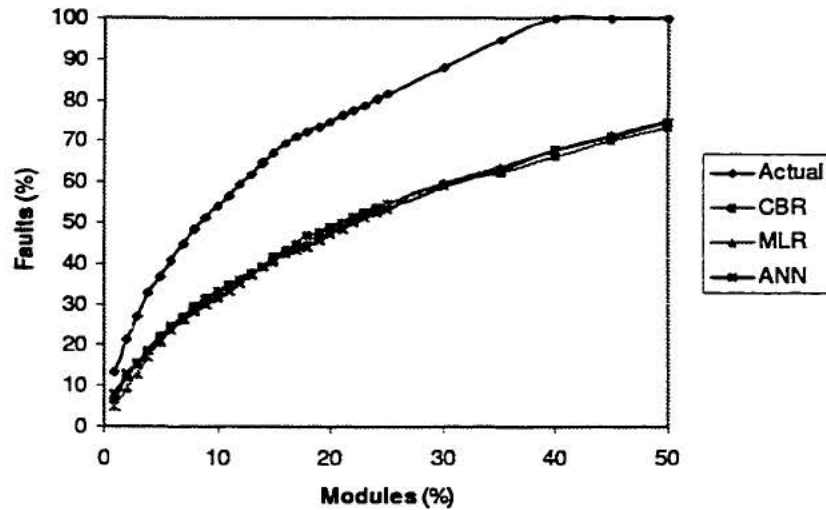


Figure 4.39: Alberg diagram for LLTS-PCA release 2: CBR, MLR, ANN diagrams.

2. *LLTS-PCA, Comparative Results for Group II*

The Alberg diagrams illustrated in Figure 4.48, 4.51 and 4.54 show that the tree-modeling methods do not generate similar ranking prediction. For release 2, group II's methods predict different ranking over both ranges I and II. For the other releases, even though they predict fairly close rankings over range I, the models diverge over range II.

For the most critical range, SPLUS obviously predicts the nearest ranking to the perfect ranking over all the ranges for release 2 and 3. This causes the performance of SPLUS to be better than the two other techniques for the critical range. For release 4, ranking based on CART-LS is the nearest to

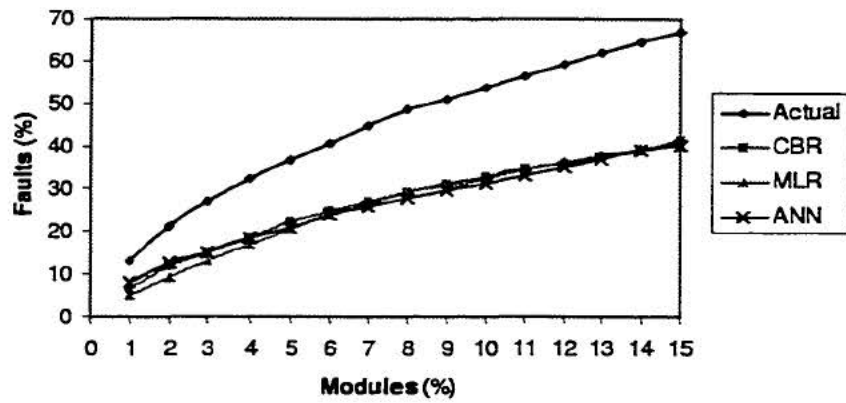


Figure 4.40: Close view of Alberg diagram for LLTS-PCA release 2: CBR, MLR, ANN

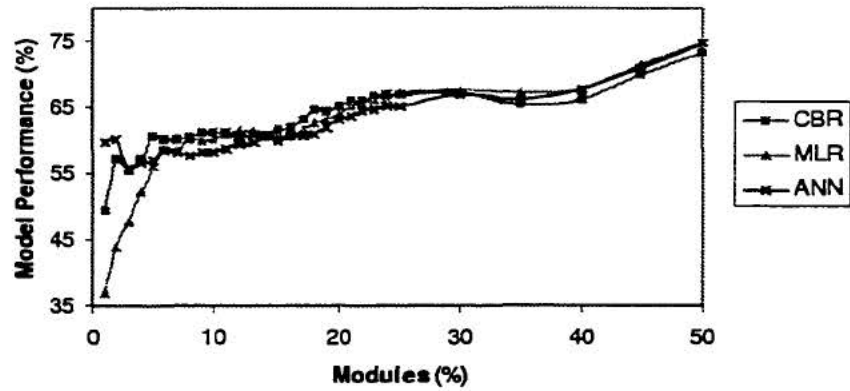


Figure 4.41: Performance of LLTS-PCA release 2: CBR, MLR, ANN

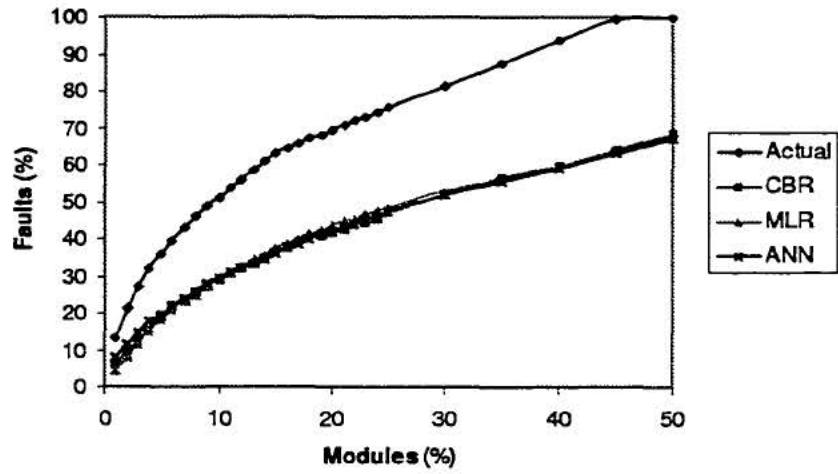


Figure 4.42: Alberg diagram for LLTS-PCA release 3: CBR, MLR, ANN

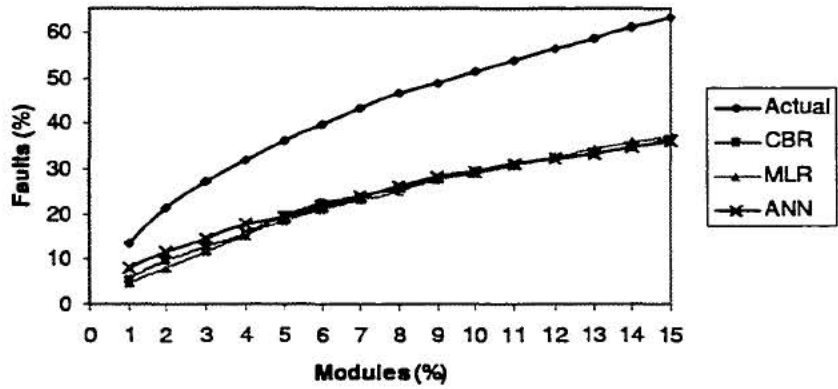


Figure 4.43: Close view of Alberg diagram for LLTS-PCA release 3: CBR, MLR, ANN

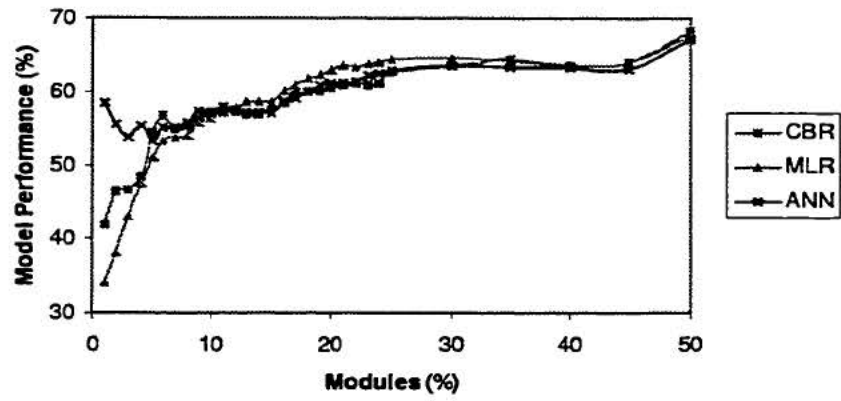


Figure 4.44: Performance of LLTS-PCA release 3: CBR, MLR, ANN

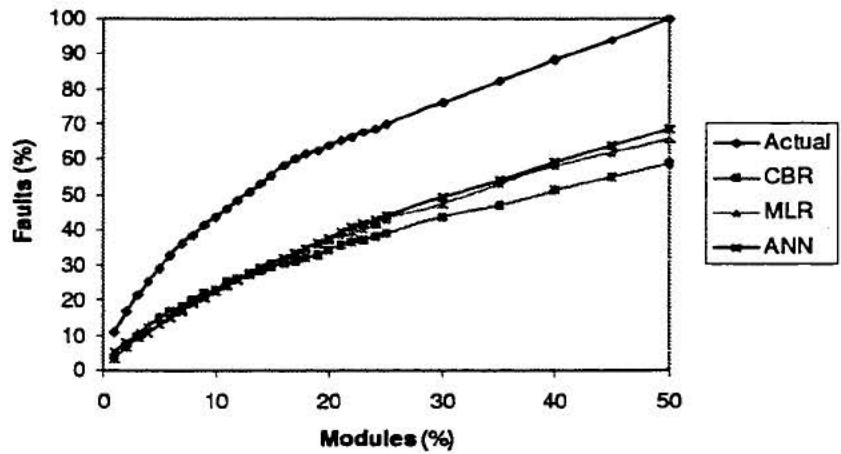


Figure 4.45: Alberg diagram for LLTS-RAW release 4: CBR, MLR, ANN

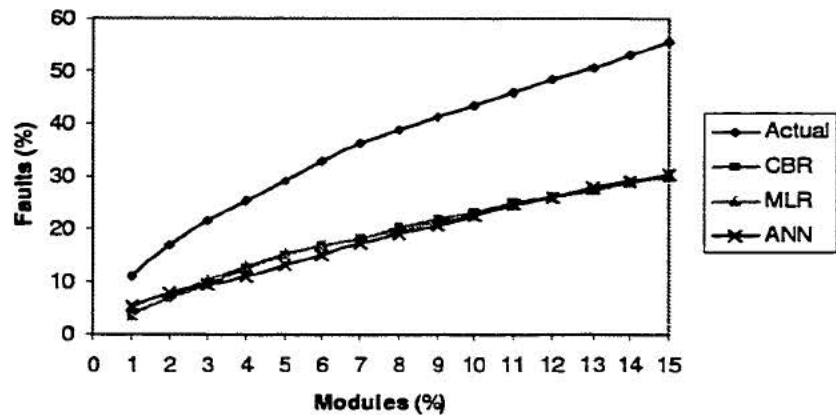


Figure 4.46: Close view of Alberg diagram for LLTS-PCA release 4: CBR, MLR, ANN

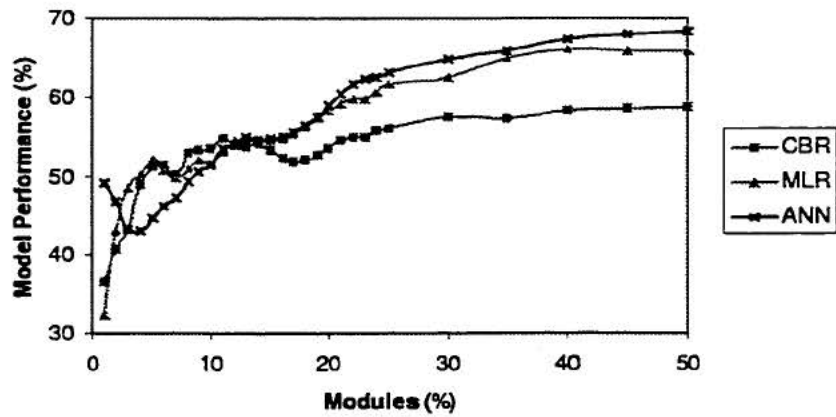


Figure 4.47: Performance of LLTS-PCA release 4: CBR, MLR, ANN

the perfect ranking for the first half of the range. The graphs are shown in Figure 4.49, 4.52 and 4.55.

When considering the performance for range *I*, SPLUS performs better than CART-LS and CART-LAD for the majority of range *I* for all releases. CART-LAD does not perform as well as other methods for the majority of this range. The performance diagrams are shown in Figure 4.50, 4.53 and 4.56.

For range *II*, the two CART methods present high variation of performances. CART-LS always shows the best performance at the end of range *II*, and CART-LAD has an inconstant trend of performance. We can observe the varying lines of CART-LAD and CART-LS in the performance diagrams for all releases. For SPLUS, this method shows fairly constant performance as for the previous range.

Concisely, CART-LS and CART-LAD generate models having high variation of $\phi(c)$ when module-order modeling. This implies that SPLUS provides more robust module-order models than CART-LAD and CART-LS for all three releases.

3. *LLTS-PCA, Group I and Group II Models Comparison*

The three techniques in group *I* present very close performances when module-order modeling. Therefore, we select one of them from each release to compare

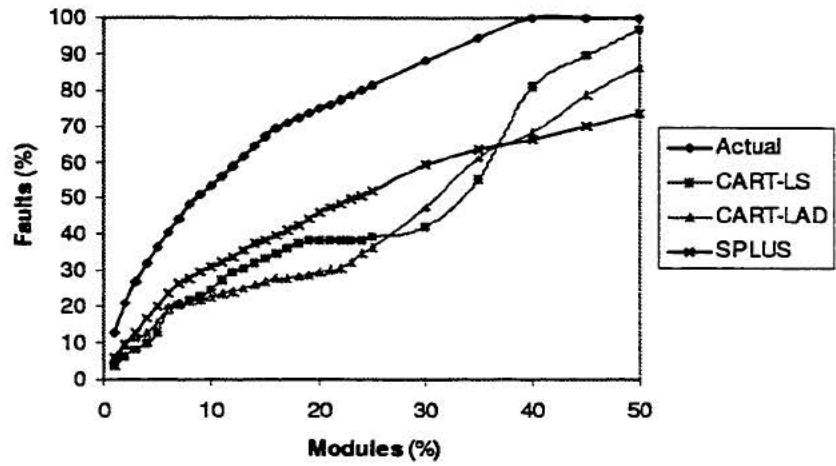


Figure 4.48: Alberg diagram for LLTS-PCA release 2: CART-LS, CART-LAD, SPLUS

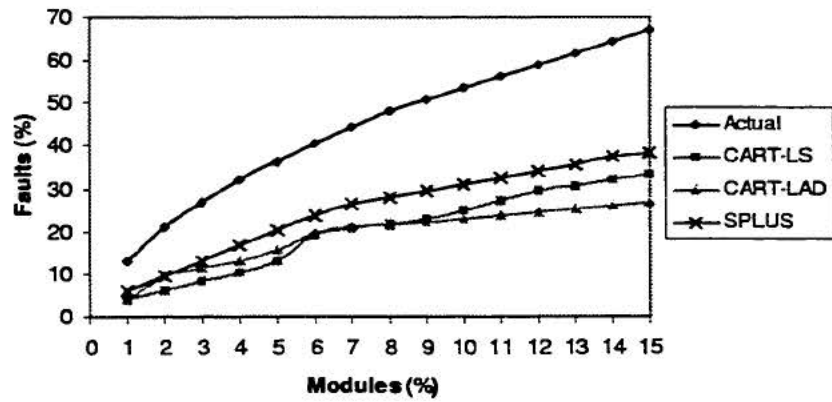


Figure 4.49: Close view of Alberg diagram for LLTS-PCA release 2: CART-LS, CART-LAD, SPLUS

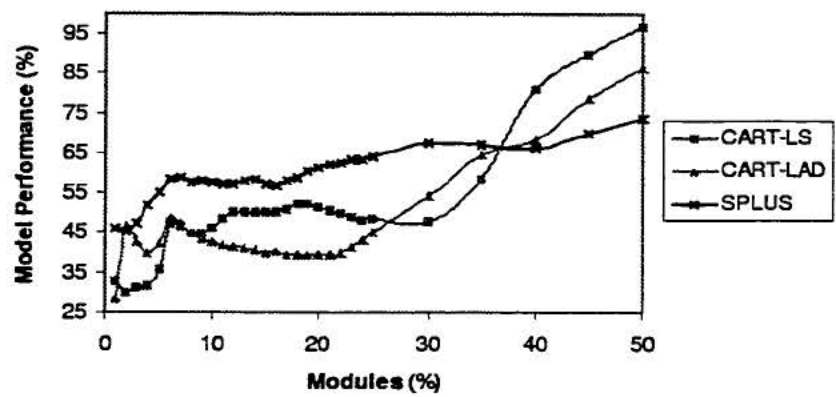


Figure 4.50: Performance of LLTS-PCA release 2: CART-LS, CART-LAD, SPLUS

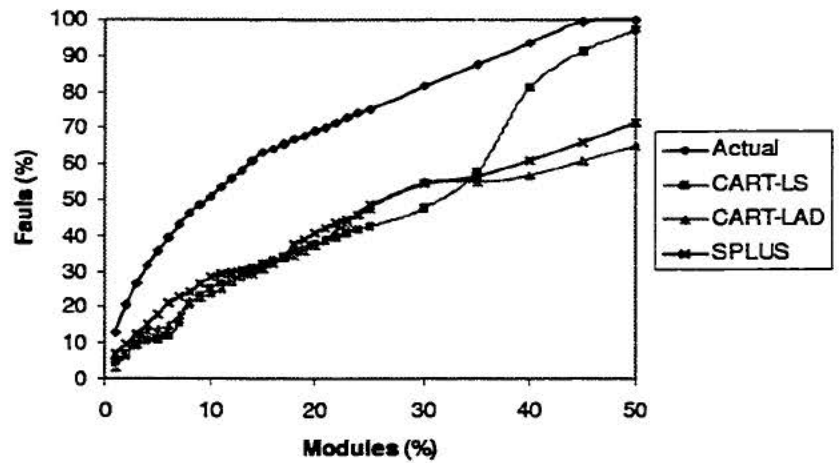


Figure 4.51: Alberg diagram for LLTS-PCA release 3: CART-LS, CART-LAD, SPLUS

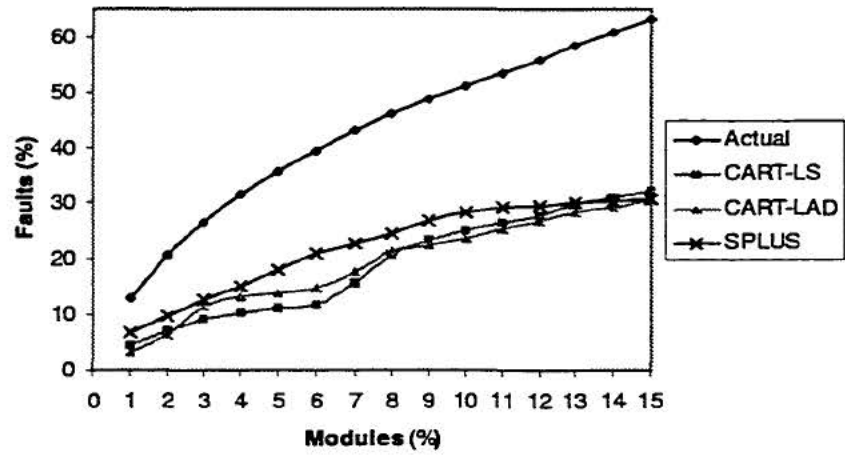


Figure 4.52: Close view of Alberg diagram for LLTS-PCA release 3: CART-LS, CART-LAD, SPLUS

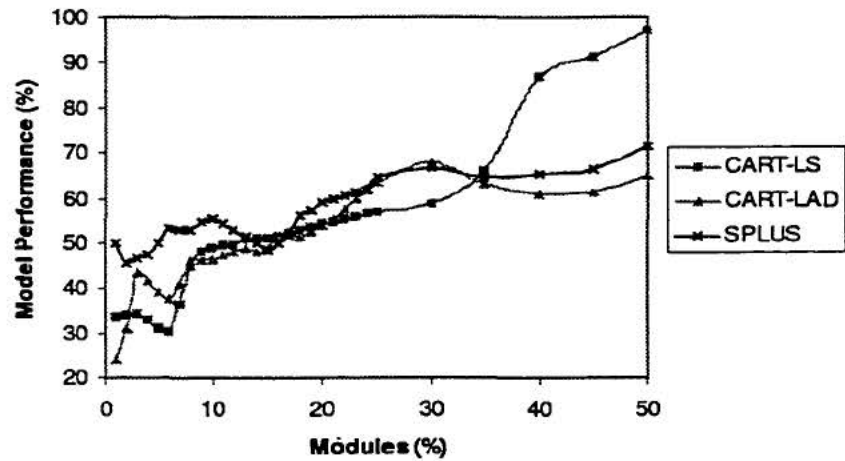


Figure 4.53: Performance of LLTS-PCA release 3: CART-LS, CART-LAD, SPLUS

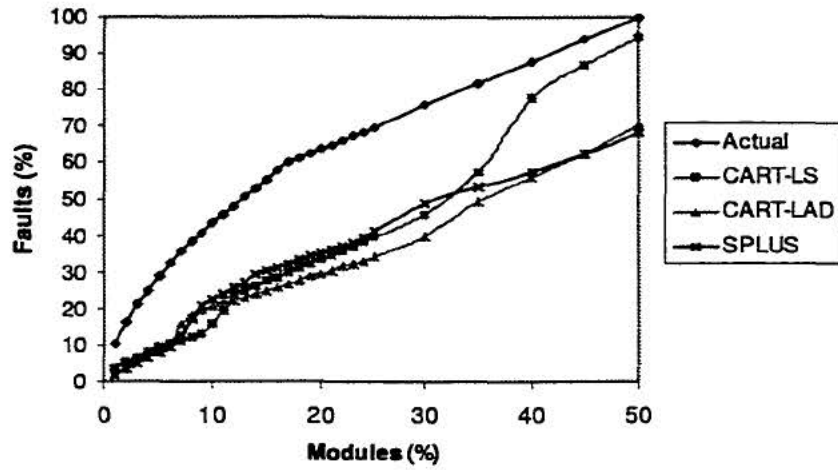


Figure 4.54: Alberg diagram for LLTS-PCA release 4: CART-LS, CART-LAD, SPLUS

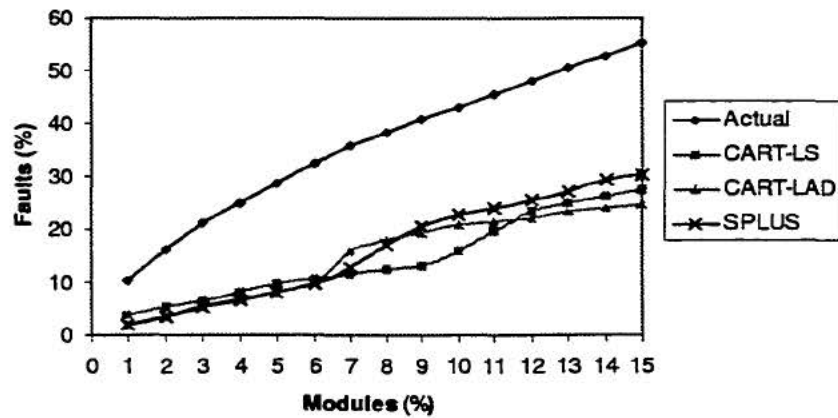


Figure 4.55: Close view of Alberg diagram for LLTS-PCA release 4: CART-LS, CART-LAD, SPLUS

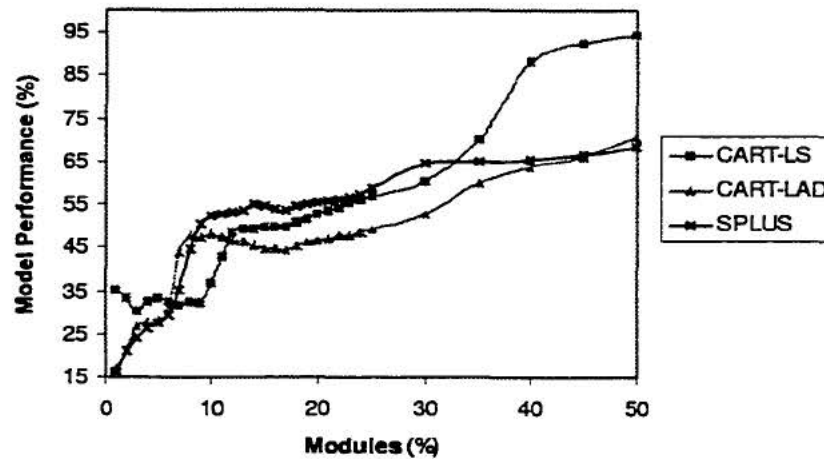


Figure 4.56: Performance of LLTS-PCA release 4: CART-LS, CART-LAD, SPLUS

with group *II*. Since ANN had a constant performance over the considered ranges for all releases, we selected ANN to represent group *I*.

The Alberg diagrams show that ANN predicts the module-ranking close to SPLUS, especially for release 2 and 4, illustrated in Figure 4.57 and 4.63. For release 3, even though some gaps existed between ANN and SPLUS depicted in Figure 4.60. ANN remains close to SPLUS compared to other techniques in group *II*.

When focusing on the most critical range, ANN predicts the ranking close to SPLUS for all releases. In addition, ANN also gave closer ranking to the perfect ranking over all the critical range than any method within group *II*, illustrated in Figure 4.58, 4.61 and 4.64. Consequently, ANN performs better than any of group *II* method over the critical range.

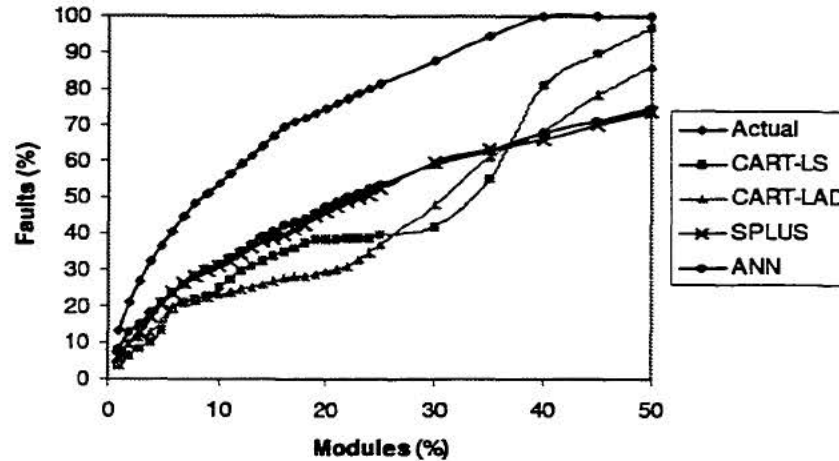


Figure 4.57: Alberg diagram for LLTS-PCA release 2: CART-LS, CART-LAD, SPLUS and ANN

When considering the performance, ANN apparently performs very close to SPLUS for release 2 and 4 over ranges *I* and *II*. For release 3, ANN performs more consistently than SPLUS, but it still shows closer performance to SPLUS than other group *II*'s methods. This indicates that SPLUS performs close to ANN, and it also implies that SPLUS performs close to the two other group *I*'s methods, CBR and MLR. The performance diagrams are shown in Figure 4.59, 4.62 and 4.65.

4. LLTS-PCA, comparative results regarding AAE and ARE

CART-LAD provided the best prediction based on AAE and ARE values for release 2 and 3, while CBR did for release 4. We compare CART-LAD to the

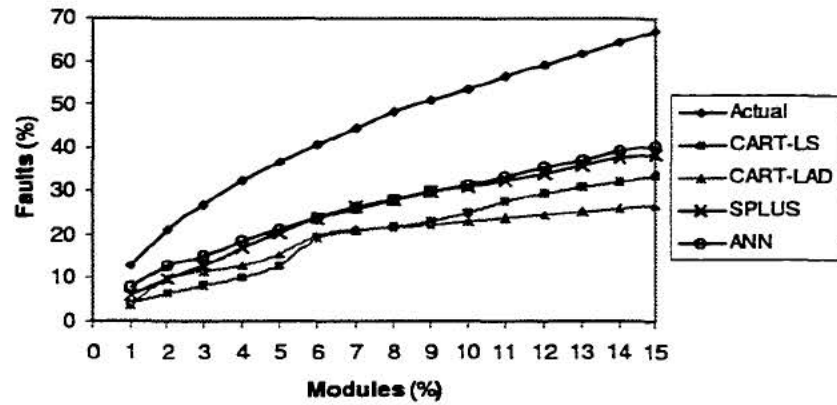


Figure 4.58: Close view of Alberg diagram for LLTS-PCA release 2: CART-LS, CART-LAD, SPLUS and ANN

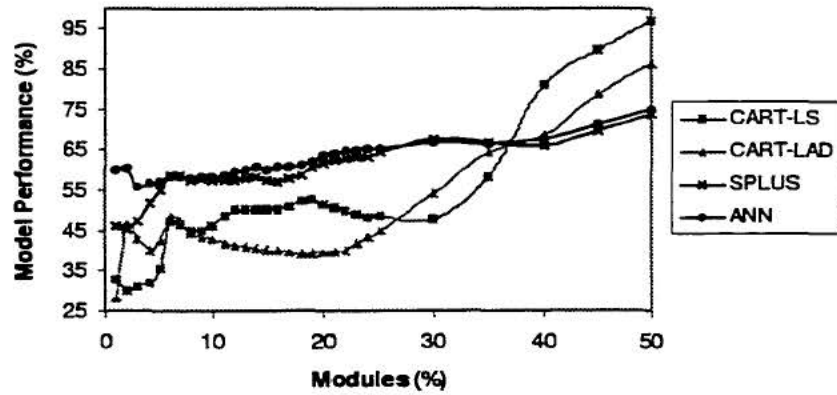


Figure 4.59: Performance of LLTS-PCA release 2: CART-LS, CART-LAD, SPLUS and ANN

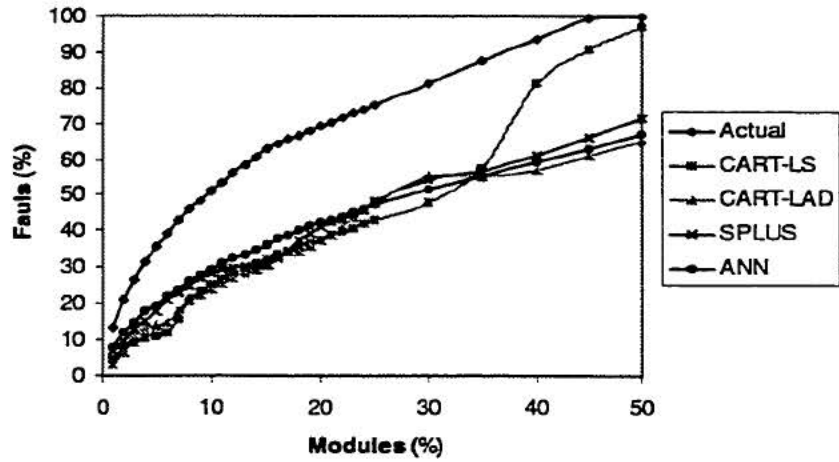


Figure 4.60: Alberg diagram for LLTS-PCA release 3: CART-LS, CART-LAD, SPLUS and ANN

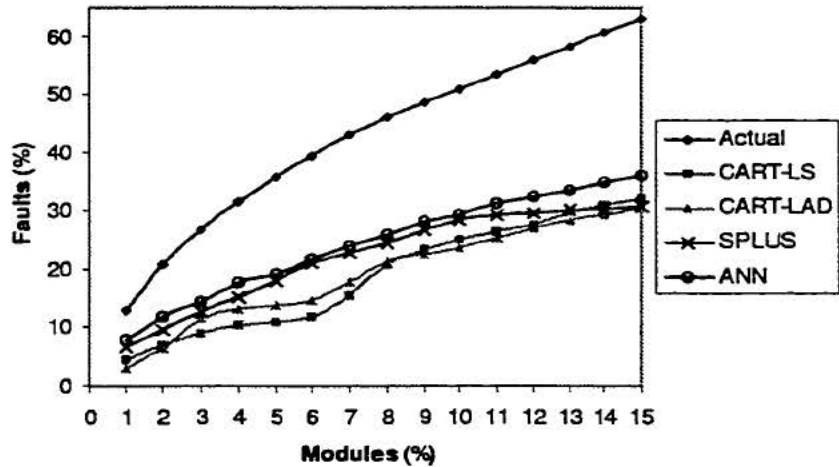


Figure 4.61: Close view of Alberg diagram for LLTS-PCA release 3: CART-LS, CART-LAD, SPLUS and ANN

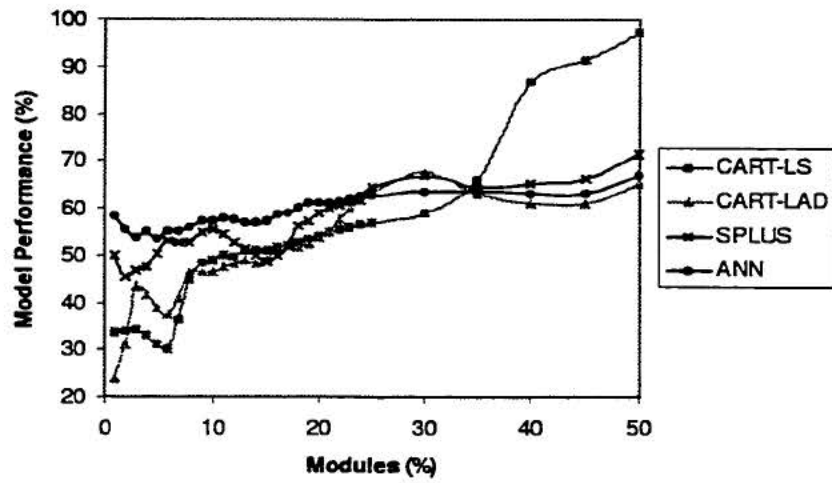


Figure 4.62: Performance of LLTS-PCA release 3: CART-LS, CART-LAD, SPLUS and ANN

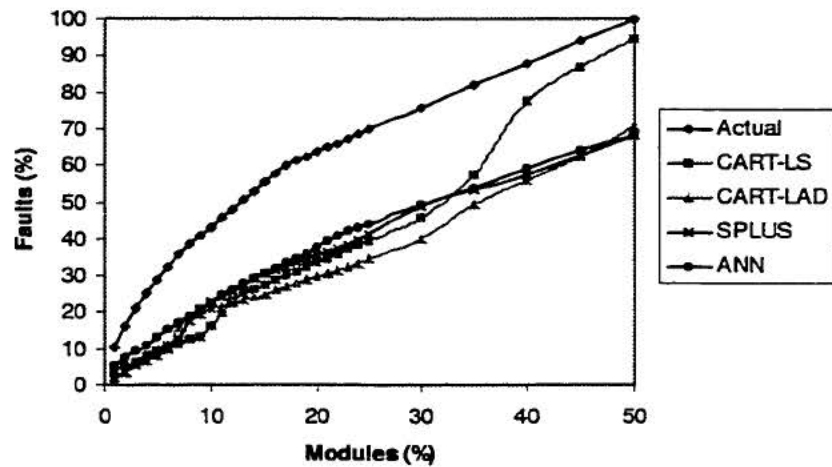


Figure 4.63: Alberg diagram for LLTS-PCA release 4: CART-LS, CART-LAD, SPLUS and ANN

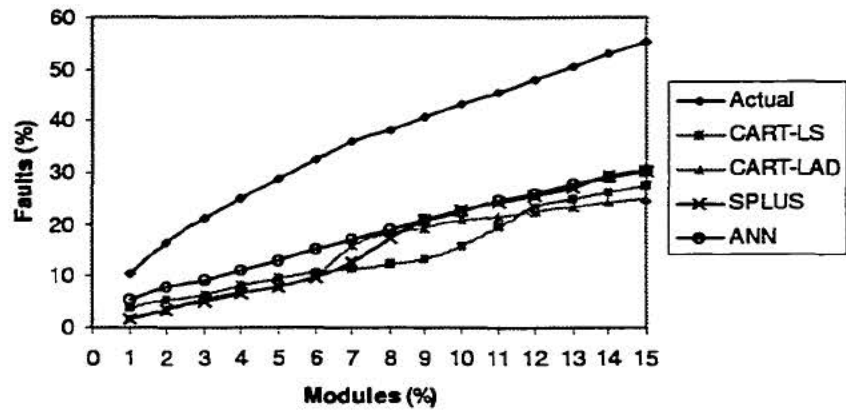


Figure 4.64: Close view of Alberg diagram for LLTS-PCA release 4: CART-LS, CART-LAD, SPLUS and ANN

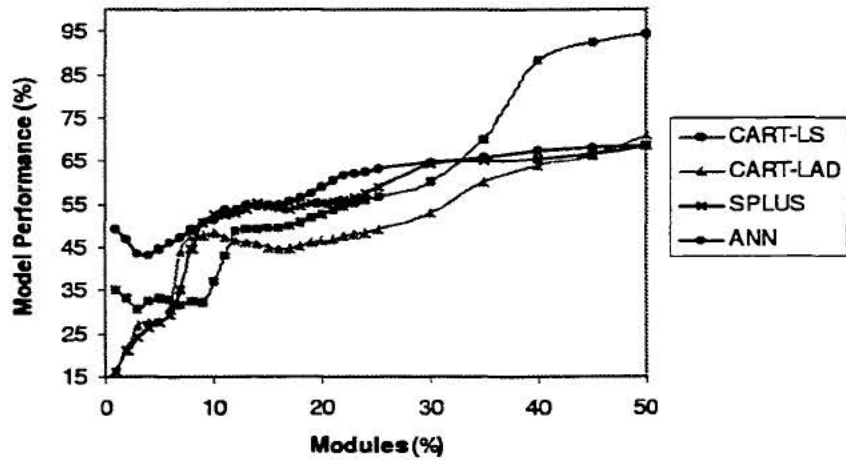


Figure 4.65: Performance of LLTS-PCA release 4: CART-LS, CART-LAD, SPLUS and ANN

techniques that bring the worst prediction for the three releases. For LLTS-PCA, CART-LS, MLR and SPLUS are the modeling techniques providing the worst prediction for release 2, 3 and 4 respectively. The graphs are depicted in Figure 4.66 - 4.74.

When considering the close view for the most critical modules, for release 2, CART-LAD presents closer ranking to the perfect ranking than CART-LS for the first half of the range (1-7 percentile). Afterwards CART-LAD does not predict a closer ranking to the perfect ranking than the other methods, illustrated in Figure 4.67. For release 3, MLR has a better ranking than CART-LAD over the critical range, depicted in Figure 4.70. For release 4, CBR gives closer ranking than SLUS before both methods gives the close ordering as shown in Figure 4.73.

When focusing on range *I* for all releases, a model based on CART-LAD apparently doesn't have as good performance compared to the other models for release 2 and 3, while CBR provided better performance than SPLUS for the first half of range *I*. For comparison on range *II*, CART-LAD and CBR do not perform as well as the compared models for the majority of the cutoff ranges, especially for release 4. The comparative performances are shown in Figure 4.68, 4.71 and 4.74.

This is a noticeable evidence that even though CART-LAD and CBR had

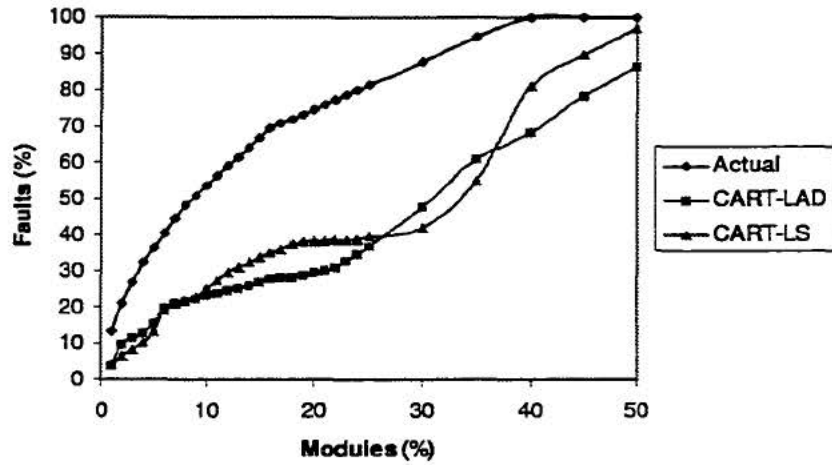


Figure 4.66: Alberg diagram for LLTS-PCA release 2: CART-LS, CART-LAD

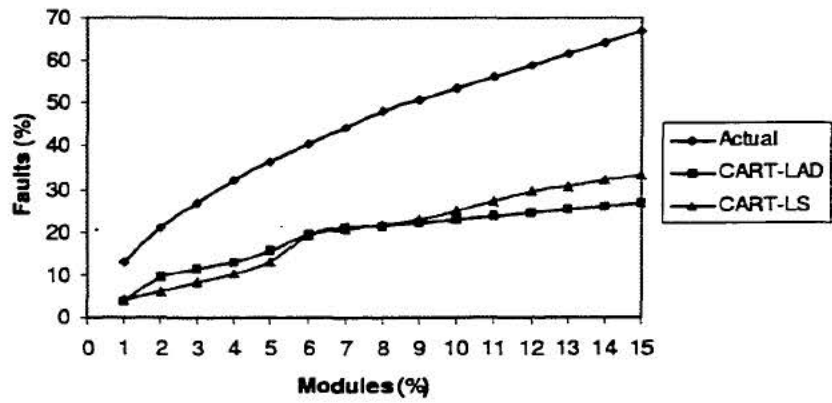


Figure 4.67: Close view of Alberg diagram for LLTS-PCA release 2: CART-LS, CART-LAD

better prediction accuracy, it did not yield better performance when module-order modeling.

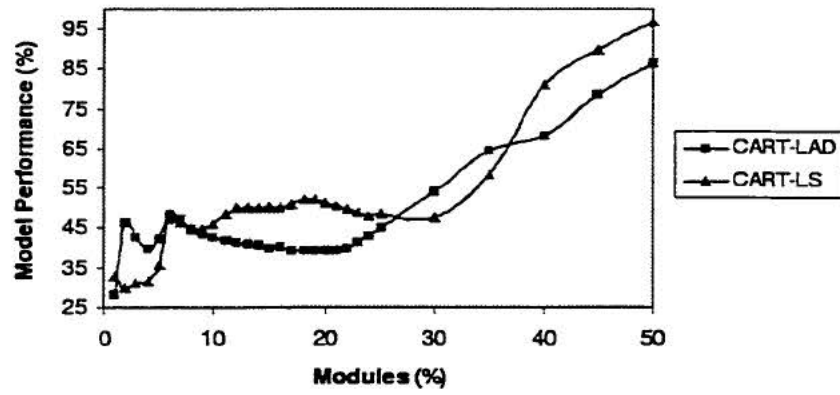


Figure 4.68: Performance of LLTS-PCA release 2: CART-LS, CART-LAD

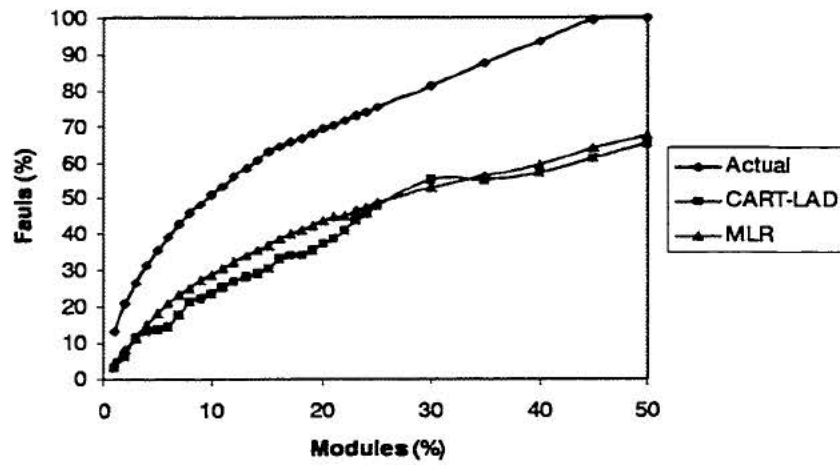


Figure 4.69: Alberg diagram for LLTS-PCA release 3: MLR, CART-LAD

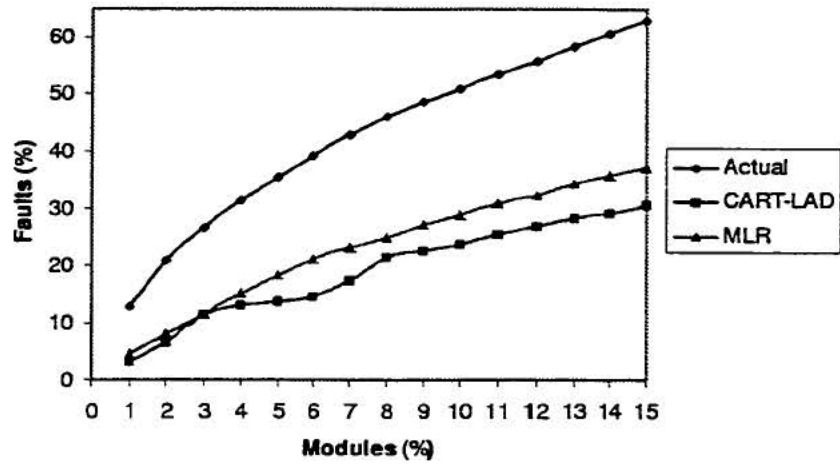


Figure 4.70: Close view of Alberg diagram for LLTS-PCA release 3: CART-LS, CART-LAD

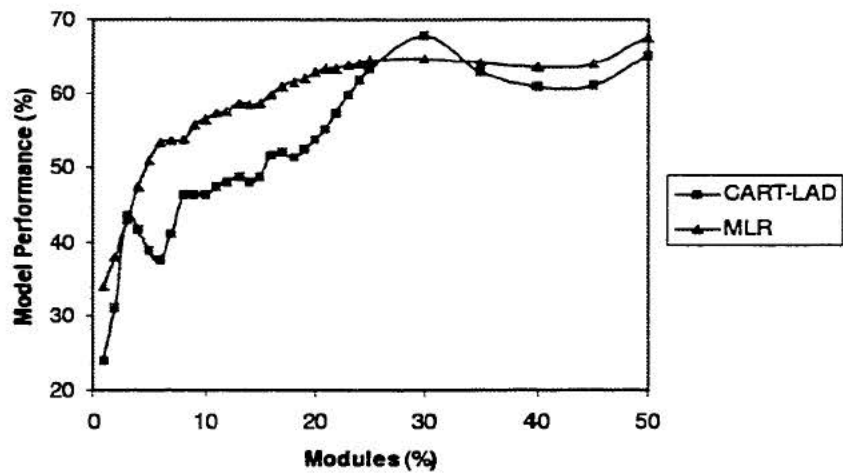


Figure 4.71: Performance of LLTS-PCA release 3: MLR, CART-LAD

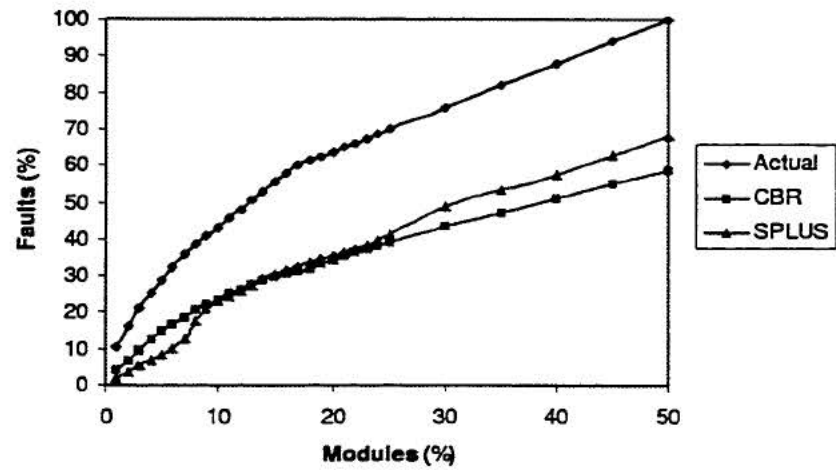


Figure 4.72: Alberg diagram for LLTS-PCA release 4: SPLUS, CBR

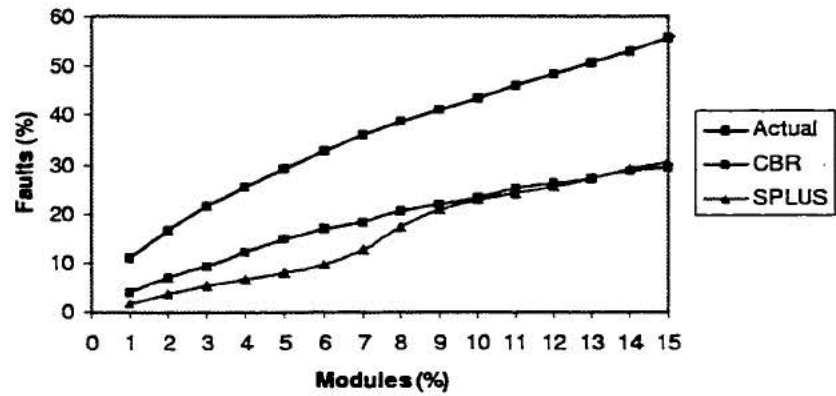


Figure 4.73: Close view of Alberg diagram for LLTS-PCA release 4: SPLUS, CBR

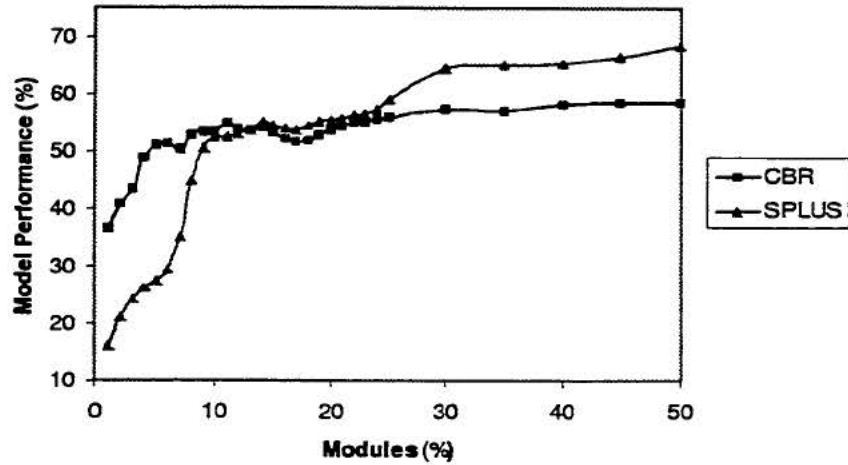


Figure 4.74: Performance of LLTS-PCA release 4: SPLUS, CBR

Table 4.8: Presentation outline for NT data

Data set	Test data set	Performance comparison
RAW	Test data set (data splitting)	(a) Group <i>I</i> (b) Group <i>II</i> (c) Group <i>I</i> and <i>II</i> (d) Best and worst prediction
PCA	Test data set (data splitting)	(a) Group <i>I</i> (b) Group <i>II</i> (c) Group <i>I</i> and <i>II</i> (d) Best and worst prediction

4.4 Experiment on NT

This section describes the module-order modeling results on the NT data set. We use the same result presentation structure as in the LLTS experiments. Table 4.5 summarizes the presentation outline for NT data.

Table 4.9: NT-RAW, Comparative accuracy of underlying quantitative models

Model	Test data	
	<i>AAE</i>	<i>ARE</i>
CBR	1.623	0.537
ANN	1.904	0.722
MLR	2.062	0.974
CART-LS	1.904	0.7618
CART-LAD	1.765	0.4632
SPLUS	1.745	0.645

4.4.1 Experiment on NT-RAW

For the NT data set, we applied the data splitting strategy to define the fit and test data sets. The models were built using the fit data set and validated using the test data set. In our study, we use test data set for module-order modeling evaluation.

- **Comparative results of the underlying quantitative models, NT-RAW**

Application of the models based on the different underlying methodologies to the NT-RAW test data set is shown in Table 4.9.

Case-Based Reasoning used respectively Mahalanobis distance and distance weighted average as case similarity function and solution algorithm for the NT-RAW data set [29].

Multiple Linear Regression used nine independent variables to build the following model.

$$\begin{aligned}
\textit{faults} = & 0.1245 + 0.0683 \cdot LP - 0.0114 \cdot UC + 0.0903 \cdot MU - 0.0023 \cdot NL \\
& + 0.0059 \cdot SPC - 0.0055 \cdot SPL + 0.0195 \cdot IFTH + 0.8363 \cdot ISNEW \\
& + 1.2716 \cdot ISCHG
\end{aligned}$$

The preferred tree using CART-LS has 5 terminal nodes and uses 3 out of 11 independent variables. For CART-LAD, the tree has 8 terminal nodes and utilizes 5 out of 11 independent variables. The selected tree by SPLUS has 28 terminal nodes and uses all 11 independent variables to build the tree [27].

For NT-RAW, the result is slightly different from the LLTS data set because CART-LAD did not give the best prediction accuracy. As presented in Table 4.9, CBR has the best prediction, regarding the least AAE and ARE values, for group *I*. For the tree-modeling group, SPLUS has the best prediction. Further, if we consider all techniques together, CBR has the best prediction and MLR has the worst.

- **Comparative results of module-order models, NT-RAW**

1. *NT-RAW, Comparative Results for Group I*

CBR and ANN perform close to each other over ranges *I* and *II*, but MLR does not have a ranking as good as the two previous models as shown in Figure 4.75.

For the most critical modules, ANN gives the farthest ranking from the perfect ranking at the beginning cutoff range (1-3 percentile). As a consequence, ANN

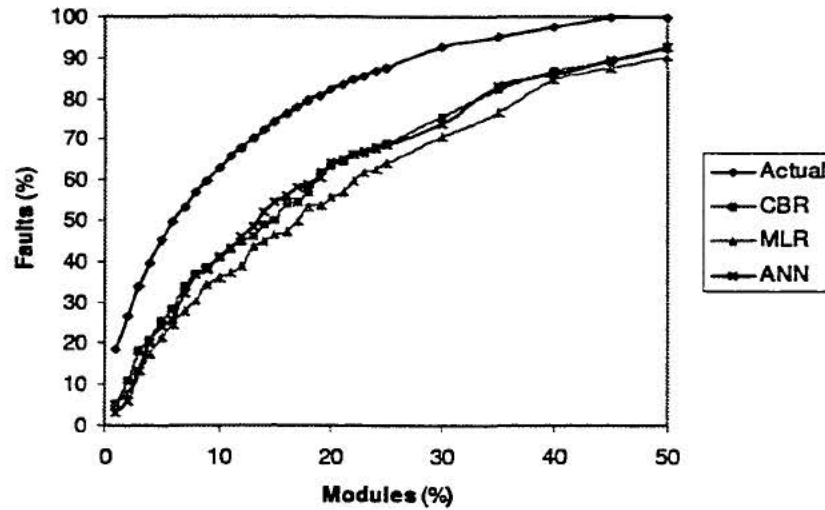


Figure 4.75: Alberg diagram for NT-RAW: CBR, MLR, ANN

does not perform as well as the two other techniques at the starting range. However, after that small range, CBR and ANN present the predicted rankings closer to the perfect ranking than MLR over the range, as shown in Figure 4.76. When focusing on the performances of the three models, CBR and ANN perform close to each other over all the considered ranges. MLR does not present an as good performance as the two previous methods for both ranges *I* and *II*, but the difference is small. The performance diagram is shown in Figure 4.77.

2. *NT-RAW, Comparative Results for Group II*

We can obviously see the difference between the rankings plotted in figure 4.78. SPLUS presents the nearest predicted ranking to the perfect ranking within group *II*. In contrast, CART-LS has the farthest prediction from the perfect

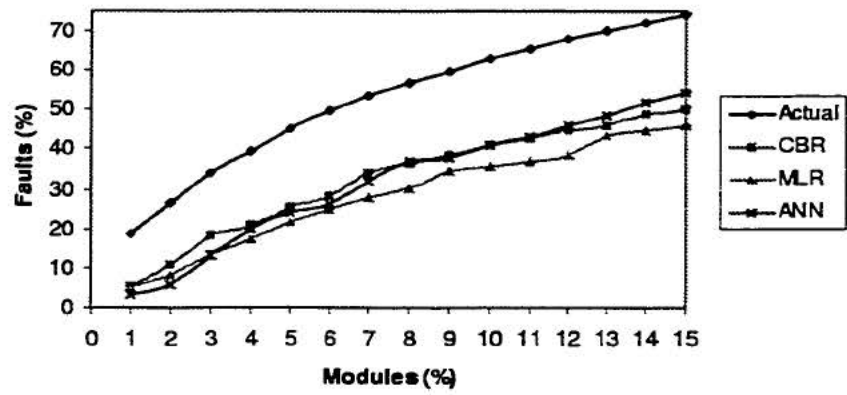


Figure 4.76: Close view of Alberg diagram for NT-RAW: CBR, MLR, ANN

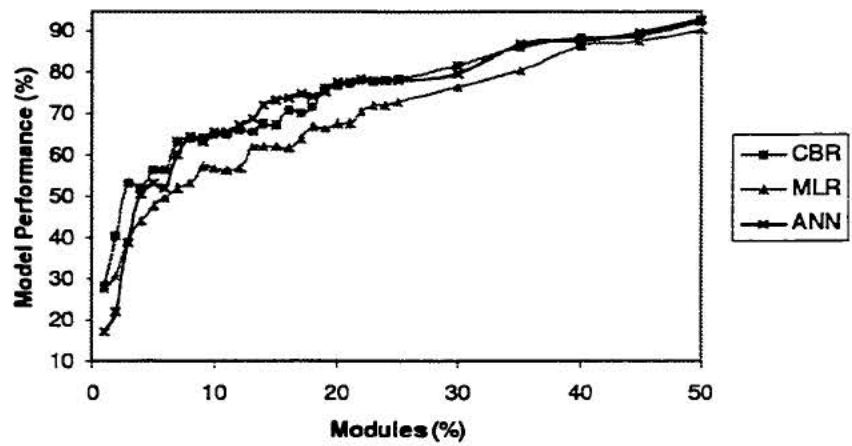


Figure 4.77: Performance of NT-RAW: CBR, MLR, ANN

ranking. A module-order model based on CART-LAD has a ranking between the two previous models.

When focusing on the most critical modules, all three methods present close predicted rankings for the starting cutoff interval (1-3 percentile). After that, SPLUS apparently provides the nearest ranking to the perfect ranking followed by CART-LAD and CART-LS respectively.

The performance illustrated in Figure 4.80 shows that the three techniques present close performances for the beginning cutoff (1-5 percentile) of range *I*. After that small range, SPLUS obviously has the best performance in group *II* followed by CART-LAD. The gap between SPLUS and CART-LAD is not large. However, the performance of CART-LS is considerably lower than the two previous models, noticeable by the large gap existing between CART-LS and those models.

When ordering the modeling techniques according to the prediction accuracy, SPLUS had the best accuracy followed by CART-LAD and CART-LS. Here the performance of the module-order models vary directly with the underlying accuracy prediction. In this case the underlying models with the best prediction accuracy also had the best performance when module-order modeling and so on.

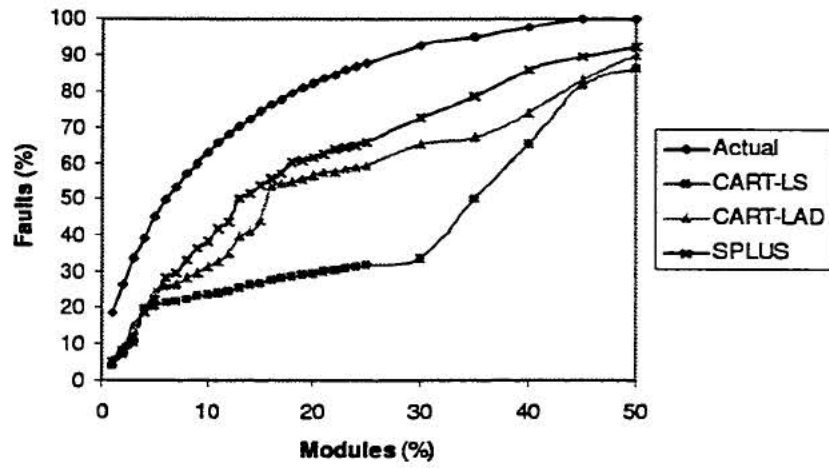


Figure 4.78: Alberg diagram for NT-RAW: CART-LS, CART-LAD, SPLUS

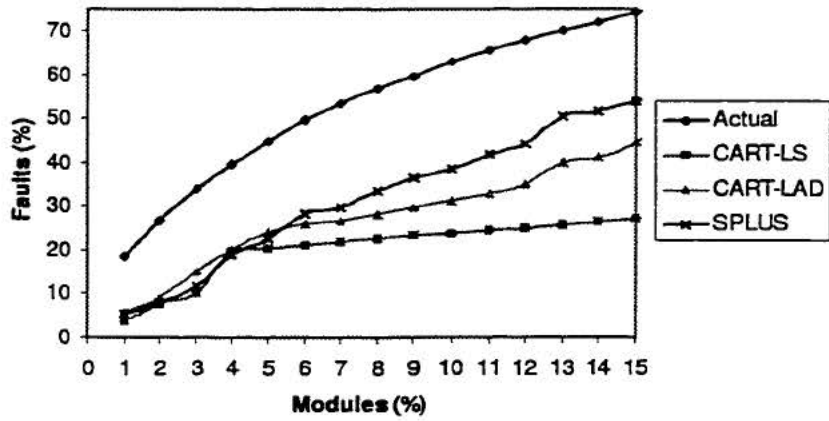


Figure 4.79: Close view of Alberg diagram for NT-RAW: CART-LS, CART-LAD, SPLUS

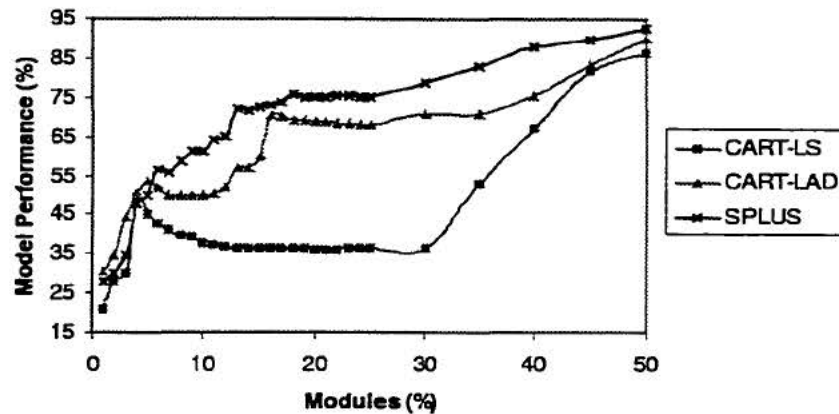


Figure 4.80: Performance of NT-RAW: CART-LS, CART-LAD, SPLUS

3. *NT-RAW, Group I and Group II Models Comparison*

We select CBR to represent group *I*. Figure 4.81, 4.82 and 4.83 show the comparative results between CBR and group *II*.

From the diagrams, we can observe that CBR performs close to SPLUS compared to CART-LS and CART-LAD. This refers that ANN and MLR also perform close to SPLUS.

4. *NT-RAW, comparative results regarding AAE and ARE*

The performance of module-order models based on CBR and MLR methods vary directly with the prediction accuracy. CBR has better prediction than MLR, and CBR performs better than MLR in module-order modeling as shown in Figure 4.84, 4.85 and 4.86.

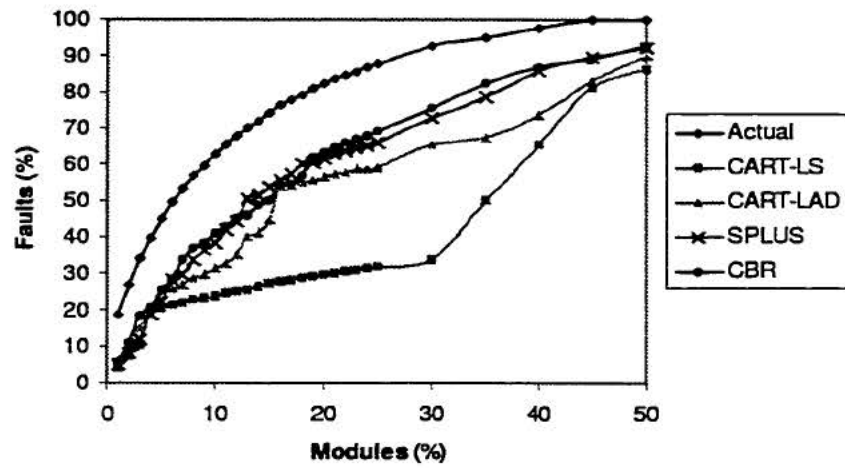


Figure 4.81: Alberg diagram for NT-RAW: CART-LS, CART-LAD, SPLUS and CBR

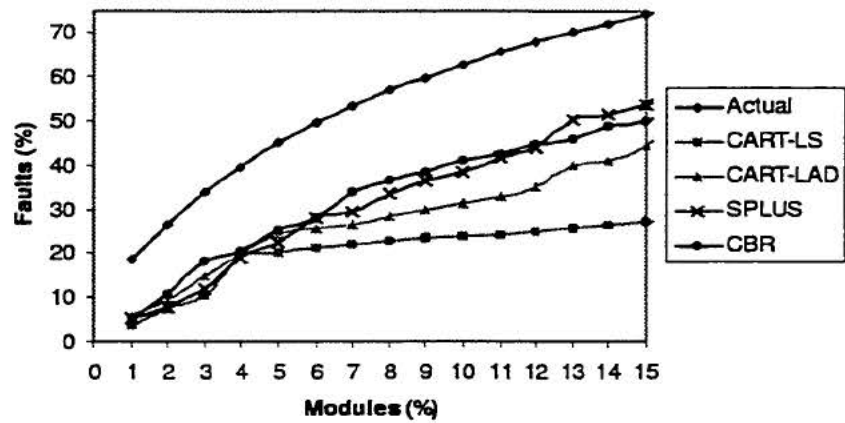


Figure 4.82: Close view of Alberg diagram for NT-RAW: CART-LS, CART-LAD, SPLUS and CBR

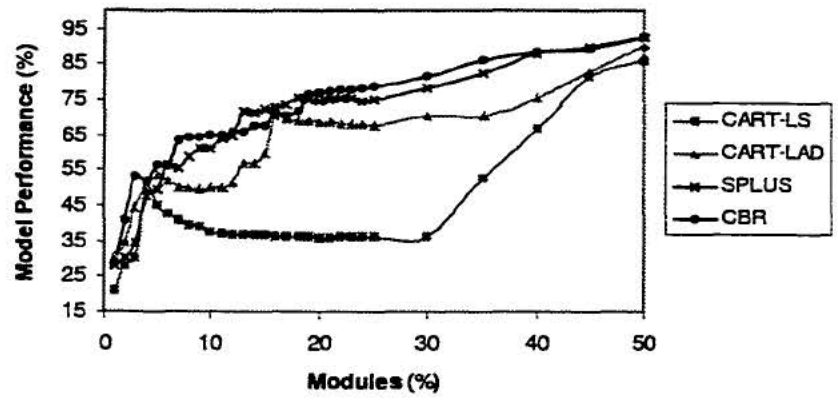


Figure 4.83: Performance of NT-RAW: CART-LS, CART-LAD, SPLUS and CBR

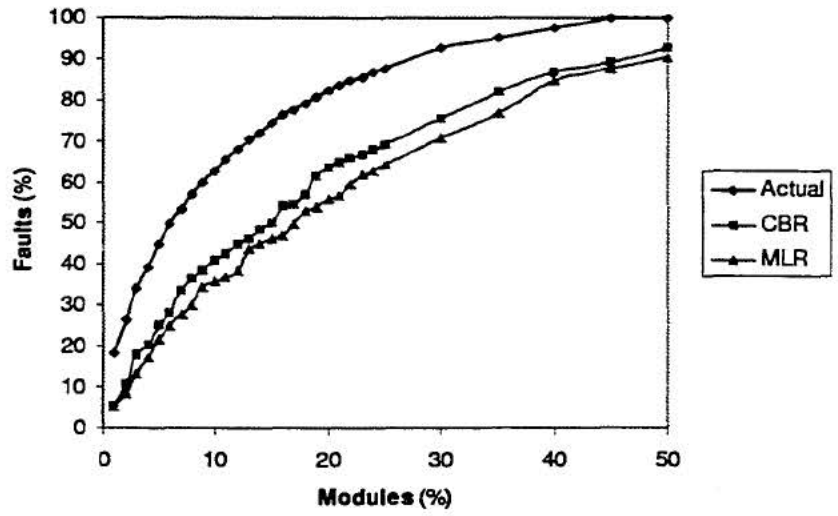


Figure 4.84: Alberg diagram for NT-RAW: CBR, MLR

This case shows that it is possible for the technique having better prediction accuracy to yield better performance when module-order modeling.

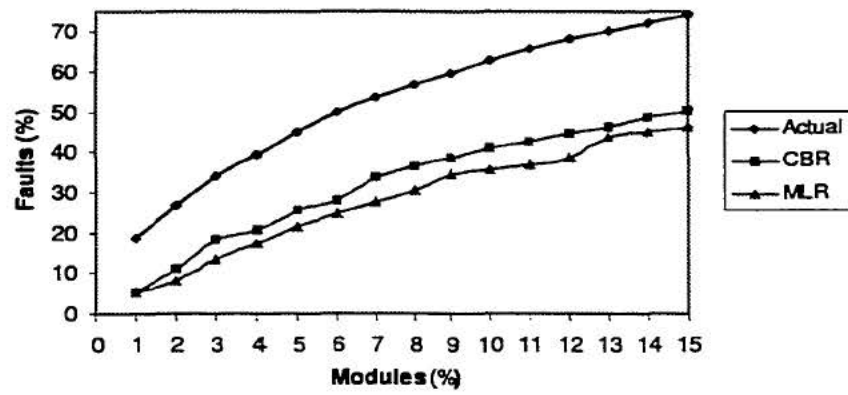


Figure 4.85: Close view of Alberg diagram for NT-RAW: CBR, MLR

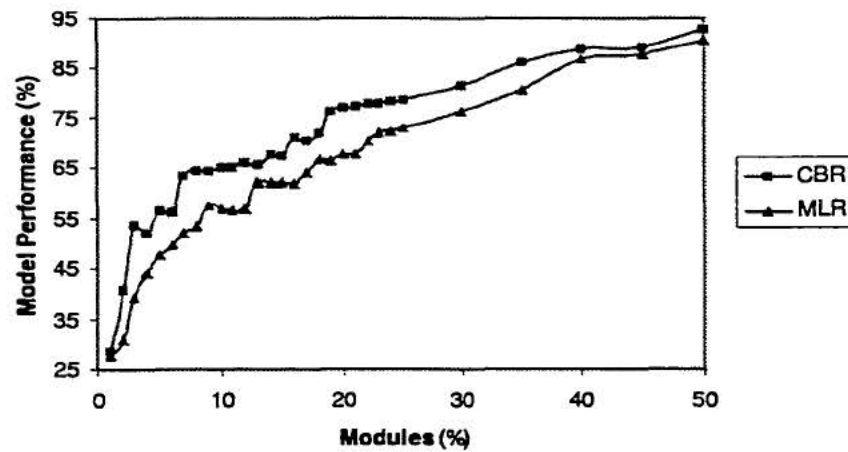


Figure 4.86: Performance of NT-RAW: CBR, MLR

Table 4.10: NT-PCA, Comparative accuracy of underlying quantitative models

Model	Test data	
	<i>AAE</i>	<i>ARE</i>
CBR	1.624	0.586
ANN	1.984	0.747
MLR	2.030	0.969
CART-LS	1.917	0.805
CART-LAD	1.607	0.376
SPLUS	1.737	0.665

4.4.2 Experiment on NT-PCA

- **Comparative results of the underlying quantitative models, NT-PCA**

Using the original data from NT-RAW, NT-PCA was obtained by applying principal components analysis. Application of the models based on different underlying methodologies to the NT-PCA's test data set is shown in Table 4.10.

The city-block distance and distance weighted average provided the best result among the available case-similarity functions and solution algorithms for CBR [29]. Using nine independent variables for multiple linear regression, the following model was obtained [29].

$$\begin{aligned} faults = & 1.1265 + 0.7854 \cdot DOMAIN1 + 0.4216 \cdot DOMAIN2 \\ & + 1.8203 \cdot DOMAIN3 + 0.8583 \cdot ISNEW + 1.3077 \cdot ISCHG \end{aligned}$$

The tree model built by CART-LS has 4 leaf nodes and uses 3 out 5 independent variables. Using CART-LAD, the tree has 5 terminal nodes and also utilizes

3 out of 5 independent variables. For SPLUS, the tree has 18 terminal nodes and uses all 5 independent variables [27].

CBR provided the best prediction in group *I* while CART-LAD presented the best prediction in group *II*. When comparing all underlying modeling methods, CART-LAD had the best prediction while MLR had the worst. The order from the best to the worst prediction for all underlying methods was CART-LAD, CBR, CART-LS, SPLUS, CART-LAD, ANN and MLR.

- **Comparative results of module-order models, NT-PCA**

1. *NT-PCA, Comparative Results for Group I*

The Alberg diagram in Figure 4.87 shows that all three techniques in this group predict close ranking models for range *I*. When considering range *II*, CBR presents a closer ranking model to the perfect ranking than the two other methods while MLR and ANN still have close rankings. However, the difference between CBR and the two other techniques is not large.

For the most critical range, the three techniques generate close ranking over the range. ANN's ranking is not as close to the perfect ranking as the two other methods after the cutoff 5 percentile, but the difference is not large as shown in Figure 4.88.

When focusing on the performance for range *I*, CBR and MLR perform relatively close for the main part of this range, while ANN performs slightly lower

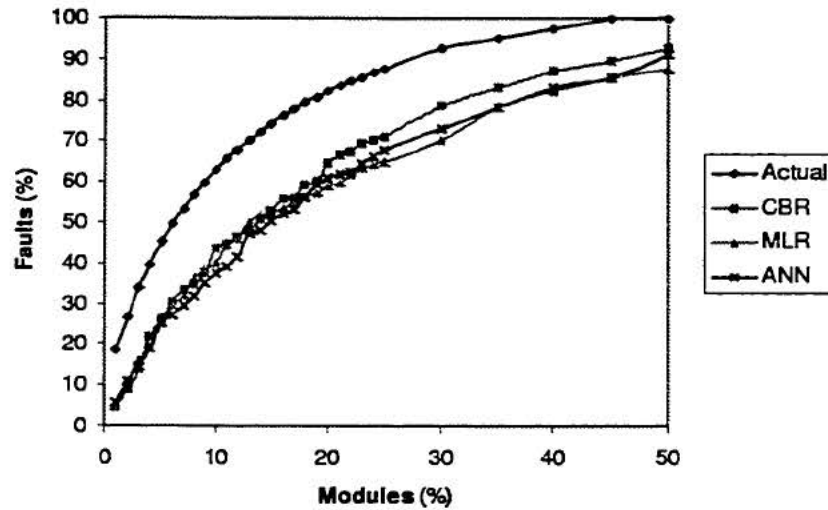


Figure 4.87: Alberg diagram for NT-PCA: CBR, MLR, ANN

than the others. For range *II*, CBR shows better performance than MLR and ANN, which present close performances along this range. Comparative performance of group *I* is illustrated in Figure 4.89.

To summarize, module-order models based on group *I* perform close to each other even though some small gaps exist along the considered ranges.

2. NT-PCA, Comparative Results for Group II

Techniques in group *II* have different ranking models for almost all the considered ranges. A predicted ranking based on SPLUS is obviously closer to the perfect ranking for both ranges *I* and *II*. The two CART methods have close module-order models for the former half of range *I*, then CART-LAD predicts a better ranking than CART-LS over the remaining ranges. The comparative

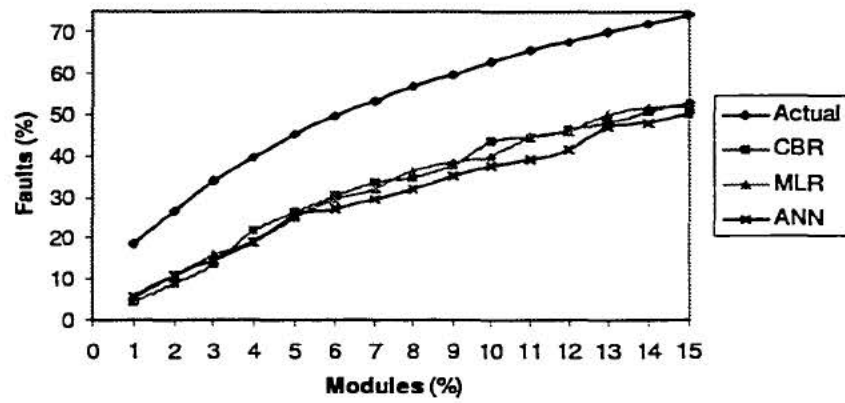


Figure 4.88: Close view of Alberg diagram for NT-PCA: CBR, MLR, ANN

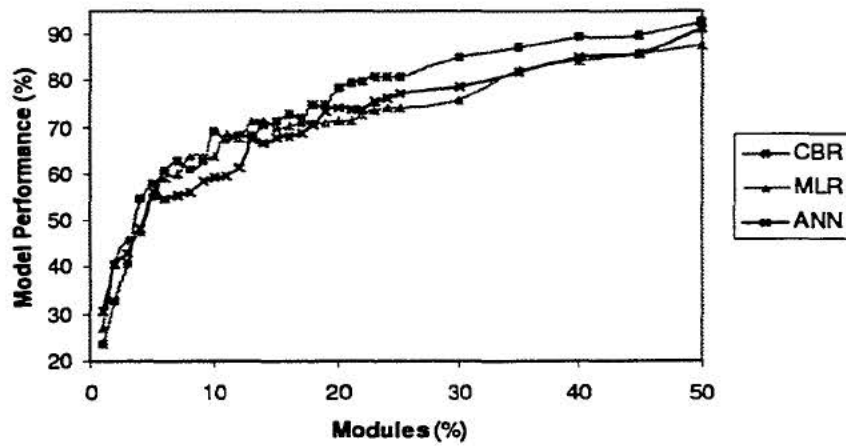


Figure 4.89: Performance of NT-PCA: CBR, MLR, ANN

rankings are shown in Figure 4.90.

When considering the most critical modules, SPLUS gives the farthest ranking from the perfect ranking at the beginning of the cutoff range (1-3 percentile), but it presents the nearest ranking after that range, shown in Figure 4.91. This causes the performance of SPLUS to be lower than the two CART techniques at the starting range, but higher afterwards.

When analyzing the performance depicted in Figure 4.92, SPLUS visibly performs better than the other two methods over the majority of range *I* and over all range *II* even though CART-LAD had better prediction accuracy than SPLUS. When considering the two CART techniques, they alternatively have higher performance for the first half of range *I* before CART-LAD outperforms CART-LS.

This case also verifies our hypothesis because CART-LAD had better prediction accuracy than SPLUS, but SPLUS presented better performance than CART-LAD when module-order modeling.

3. *NT-PCA, Group I and Group II Models Comparison*

Since MLR performs close to CBR and ANN for ranges *I* and *II* respectively, we select MLR to represent group *I*. The comparative results are plotted in Figure 4.93 and 4.95.

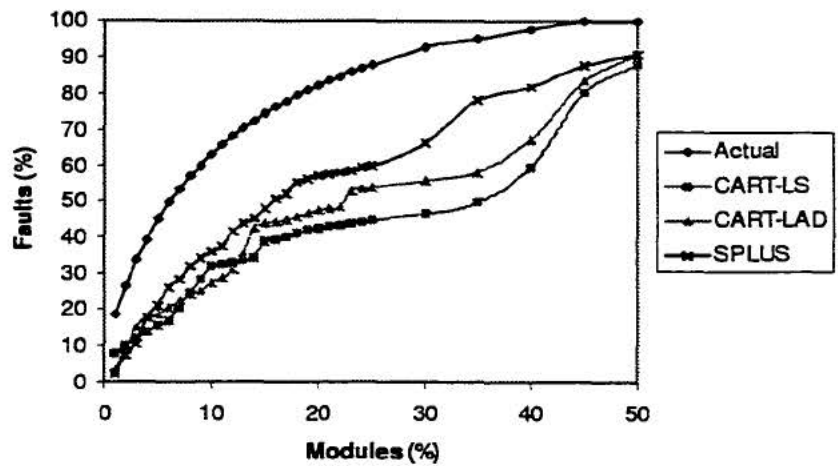


Figure 4.90: Alberg diagram for NT-PCA: CART-LS, CART-LAD, SPLUS

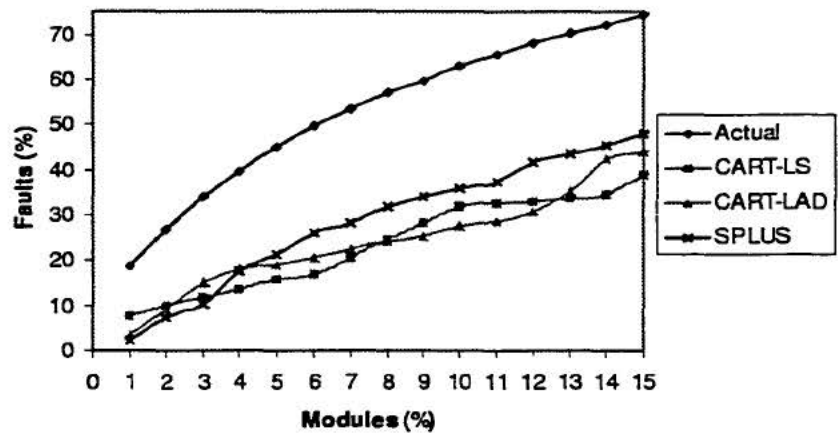


Figure 4.91: Close view of Alberg diagram for NT-PCA: CART-LS, CART-LAD, SPLUS

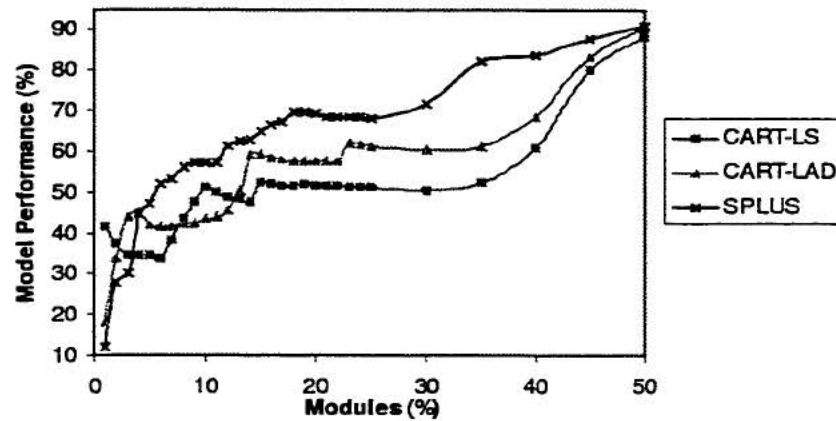


Figure 4.92: Performance of NT-PCA: CART-LS, CART-LAD, SPLUS

The Alberg diagram in Figure 4.93 shows that MLR presents predicted ranking closer to the perfect ranking than all techniques in group *II*. When comparing MLR with group *II* methods, a module-order model based on MLR is close to SPLUS. This implies that SPLUS also has a close ranking compared to CBR and ANN. In addition, the three techniques in group *I* predicts the ranking closer to the perfect ranking than all group *II* methods.

For the close view of the most critical modules, MLR presents closer ranking to the perfect ranking than all group *II* methods except for the first cutoff percentile, illustrated in Figure 4.94. As a consequence, MLR's performance is lower than CART-LS's only for the first percentile, but obviously higher than all methods of group *II* afterwards.

For the performances, it is observed that MLR performs better than all group *II*'s methods over range *I* and *II*. When comparing MLR's performance with

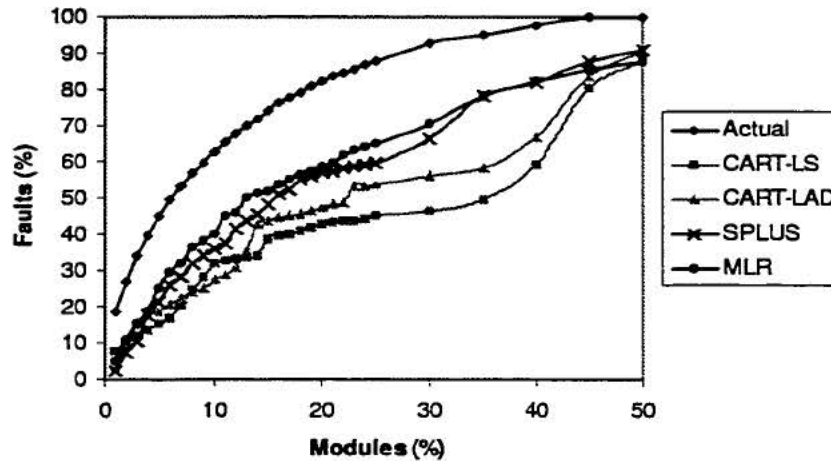


Figure 4.93: Alberg diagram for NT-PCA: CART-LS, CART-LAD, SPLUS and MLR

group *II*, MLR performs close to SPLUS over the main part of the considered ranges *I* and *II* compared to the two CART methods. This infers that group *I*'s methods also have higher performance than all group *II*'s techniques for the majority of the ranges even though CART-LAD had a better prediction accuracy than all group *I*'s methods.

4. *NT-PCA, comparative results regarding AAE, ARE*

CART-LAD and MLR had the best and the worst quantitative prediction. However, the graphs illustrated in Figure 4.96, 4.97 and 4.98 apparently present that MLR provides a closer predicted ranking to the perfect ranking than CART-LAD and has a higher performance than CART-LAD when

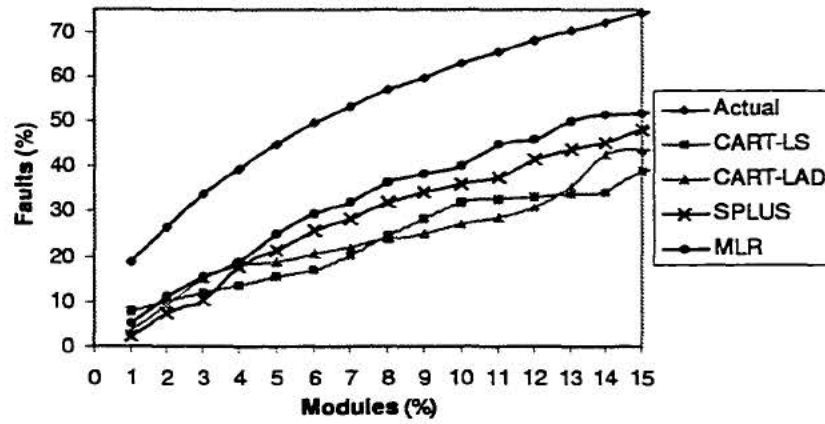


Figure 4.94: Close view of Alberg diagram for NT-PCA: CART-LS, CART-LAD, SPLUS and MLR

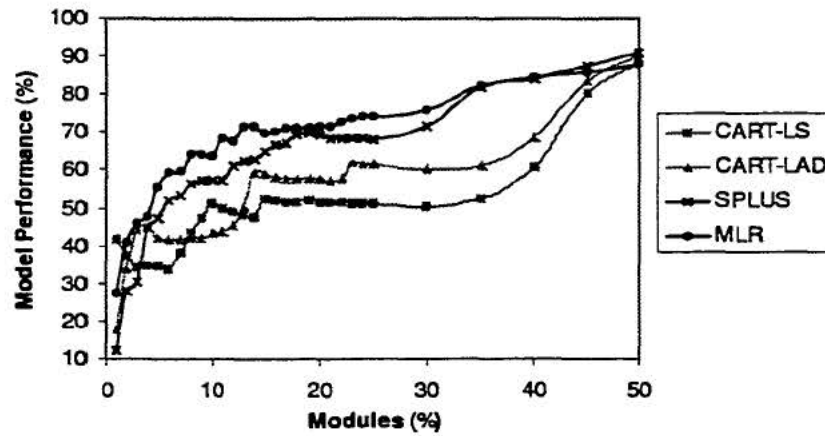


Figure 4.95: Performance of NT-PCA: CART-LS, CART-LAD, SPLUS and MLR

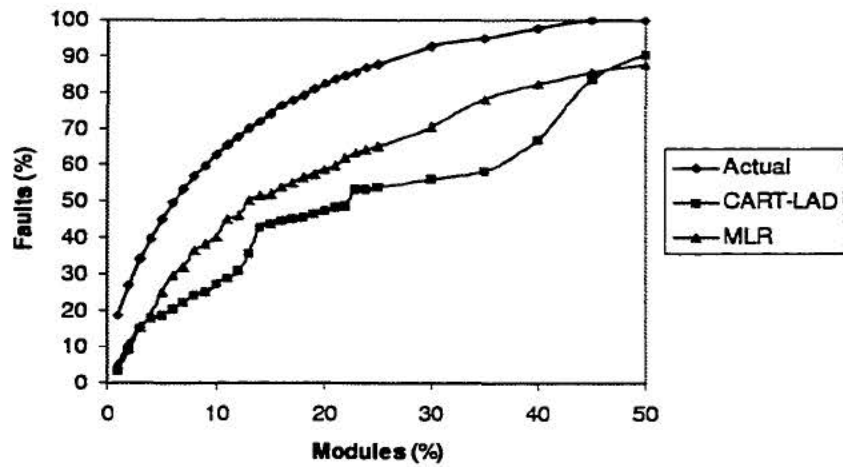


Figure 4.96: Alberg diagram of NT-PCA: MLR, CART-LAD

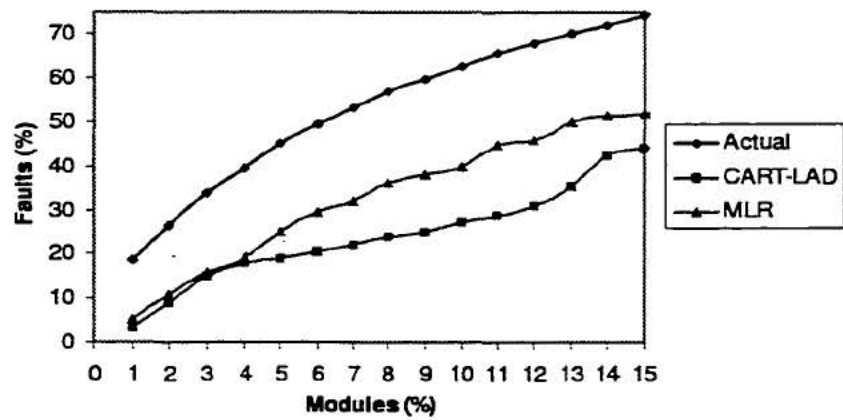


Figure 4.97: Close view of Alberg diagram for NT-PCA: MLR, CART-LAD

module-order modeling over all the considered ranges. This repeatedly confirms our hypothesis like most of the previous cases.

4.5 Experiment on LNTS

Table 4.11 summarizes the presentation outline for LNTS data.

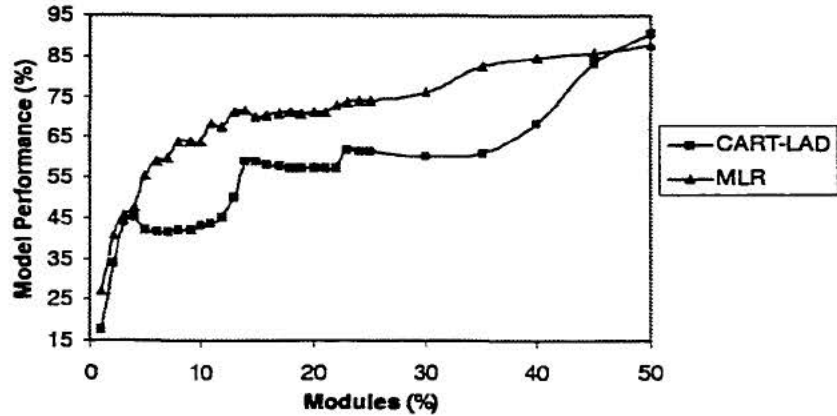


Figure 4.98: Performance of NT-PCA: MLR, CART-LAD

Table 4.11: Presentation outline for LNTS data

Data set	Test data set	Performance comparison
RAW	Test data set (data splitting)	(a) Group <i>I</i> (b) Group <i>II</i> (c) Group <i>I</i> and <i>II</i> (d) Best and worst prediction
PCA	Test data set (data splitting)	(a) Group <i>I</i> (b) Group <i>II</i> (c) Group <i>I</i> and <i>II</i> (d) Best and worst prediction

4.5.1 Experiment on LNTS-RAW

LNTS data also used the data splitting technique to define the fit and test data sets as done in NT data. As usual, the fit data set is used to build the model using five different underlying quantitative methodologies. The test data set is used to validate and compare the accuracy of the prediction models. In addition, it is also used to validate the module-order models.

Table 4.12: LNTS-RAW, Comparative accuracy of underlying quantitative models

Model	Test data	
	<i>AAE</i>	<i>ARE</i>
CBR	1.169	0.546
ANN	1.284	0.665
MLR	1.263	0.667
CART-LS	1.2894	0.6853
CART-LAD	1.131	0.376
SPLUS	1.2889	0.6845

• **Comparative results of the underlying quantitative models, LNTS-RAW**

We applied the test data to the underlying methods, the results are shown in Table 4.12.

Case-Based Reasoning used the Mahalonobis distance and distance weighted average as similarity function and solution algorithm since they provided the best accuracy for LNTS-RAW [29].

After using stepwise selection (5% significance level), 5 out of 9 independent variables were chosen to build the following Multiple Linear Regression model [29].

$$\begin{aligned} faults = & 0.60812866 - 0.00110394 \cdot TCT + 0.01106293 \cdot VG - 0.01854467 \cdot NL \\ & + 0.00082968 \cdot IFTH - 0.0017737 \cdot NELTOT \end{aligned}$$

The tree built using CART-LS has 3 terminal nodes and uses 3 out of 9 independent variables. The tree built using CART-LAD has 4 leaf nodes and uses 2 out of 9 independent variables. The tree built using SPLUS has 16 leaf [27].

When comparing group *I* models, CBR has better prediction accuracy than MLR and ANN. For the tree modeling group, CART-LAD gave better prediction than CART-LS and SPLUS. The accuracy of CART-LS and SPLUS are almost identical. Further, when comparing all five underlying quantitative models, the order from the best to the worst prediction is CART-LAD, CBR, MLR, ANN, SPLUS and CART-LS.

- **Comparative results of module-order models, LNTS-RAW**

1. *LNTS-RAW, Comparative Results for Group I*

Module-order models based on the three techniques are very close to each other over all the considered ranges as shown in Figure 4.99 and 4.100.

When considering performance for range *I*, the three methods present very close performances. MLR only performs slightly better than the other methods in the cutoff range, 5-20. When focusing on range *II*, CBR has a slightly better performance than the other two methods at the cutoff 35-50, while MLR and ANN perform very close to each other. The comparative performance are plotted in Figure 4.101.

2. *LNTS-RAW, Comparative Results for Group II*

The Alberg diagram in Figure 4.102 shows that SPLUS predicts the ranking closer to the perfect ranking than the two other methods over most of range *I*. For range *II*, CART-LAD and SPLUS have very close predicted rankings,

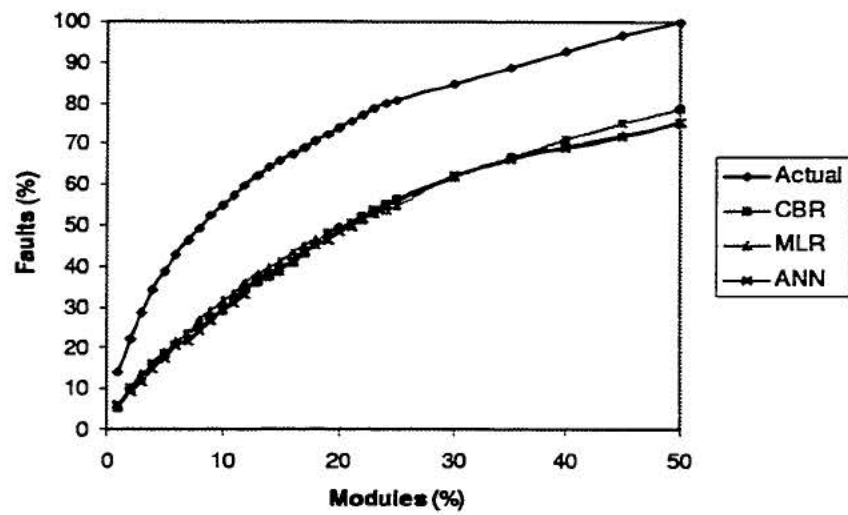


Figure 4.99: Alberg diagram for LNTS-RAW: CBR, MLR, ANN

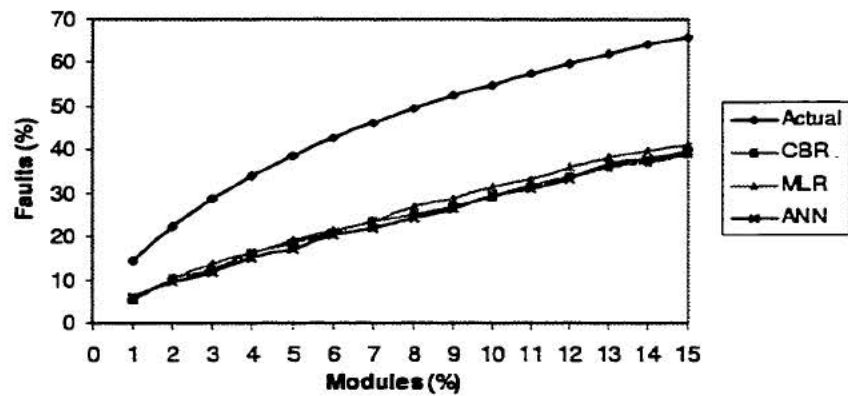


Figure 4.100: Close view of Alberg diagram for LNTS-RAW: CBR, MLR, ANN

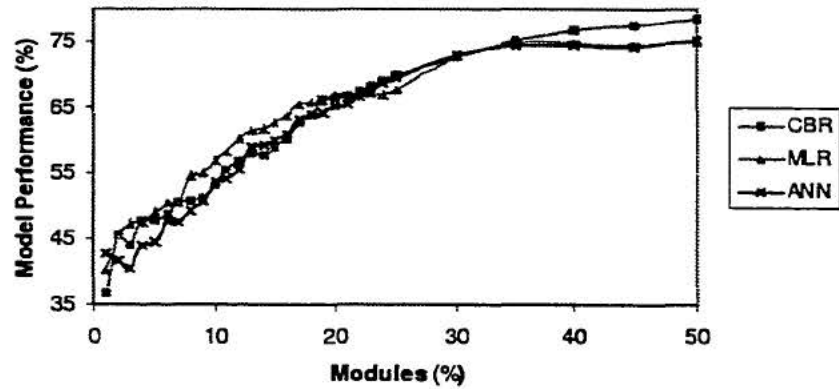


Figure 4.101: Performance of LNTS-RAW: CBR, MLR, ANN

while CART-LS does not predict the ranking as close to the perfect ranking as the other techniques for the first half of range *II*. However, CART-LS provides the predicted ranking visibly closer to the perfect ranking than CART-LAD and SPLUS in the second half of range *II*.

When focusing on the most critical modules, SPLUS presents closer ranking to the perfect ranking than the two CART techniques except at the first cutoff percentile and at the end of the critical range as shown in Figure 4.103.

The module-order modeling performance using SPLUS is better than for the two CART techniques for range *I*, except for a small interval of cutoff (12-16 percentile), while the two CART methods have high variation of $\phi(c)$ compared to SPLUS for range *I*. When considering range *II*, SPLUS and CART-LAD have close performances, while CART-LS has an inconstant trend of performance along this range. CART-LS performs visibly not as good as the other

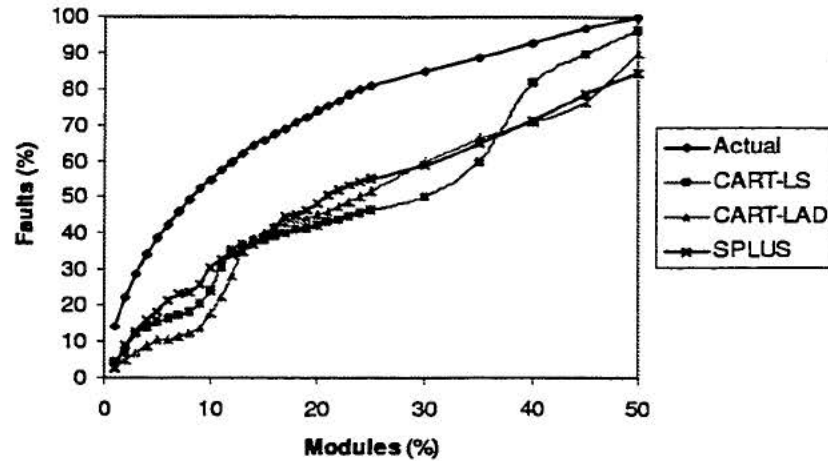


Figure 4.102: Alberg diagram for LNTS-RAW: CART-LS, CART-LAD, SPLUS

two methods for the first half, but it evidently has higher performance for the second half of range *II*. The comparative performance is depicted in Figure 4.104.

To summarize, SPLUS seems to have less variation of $\phi(c)$ than the other models over both ranges. This states that a module-order model based on SPLUS is more robust than those based on the two CART techniques.

3. LNTS-RAW, Group I and Group II Models Comparison

We chose CBR to be a representative of group *I*. The Alberg diagram in Figure 4.105 denotes that CBR predicts the ranking model very close to SPLUS over all the considered ranges. Both methods almost predict identical ranking models over range *I*, and the two models have some small difference over

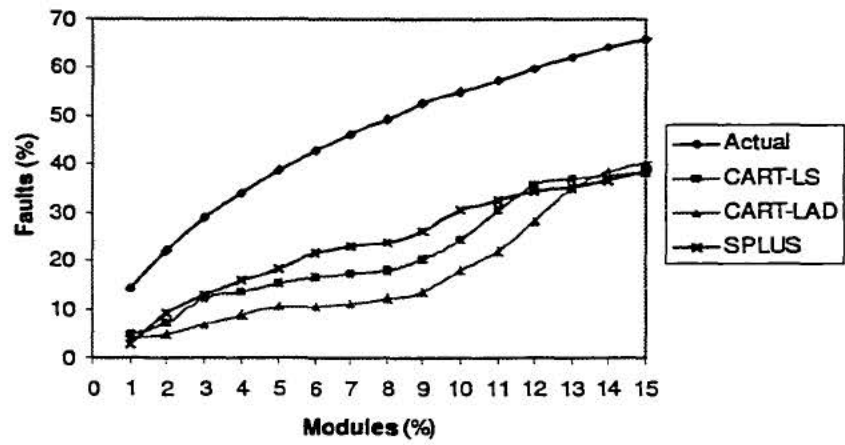


Figure 4.103: Close view of Alberg diagram for LNTS-RAW: CART-LS, CART-LAD, SPLUS

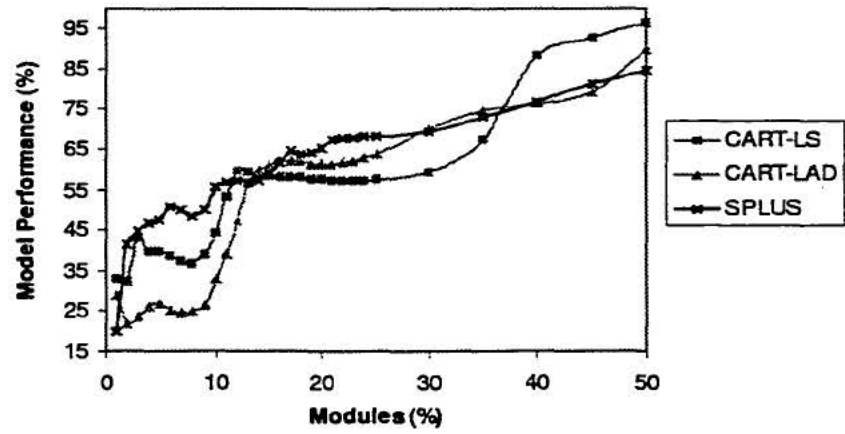


Figure 4.104: Performance of LNTS-RAW: CART-LS, CART-LAD, SPLUS

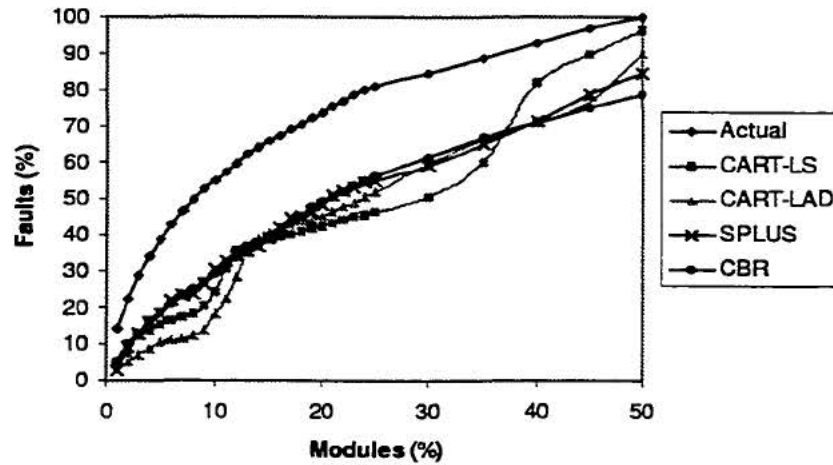


Figure 4.105: Alberg diagram for LNTS-RAW: CART-LS, CART-LAD, SPLUS and CBR

range *II*. The same results are noticed when we take a close view to the most critical range as shown in Figure 4.106.

When focusing on the performances shown in Figure 4.107, SPLUS almost has the same performance as CBR over range *I*. Considering range *II*, they still perform close to each other compared to other methods. This means that SPLUS also perform close to MLR and ANN.

4. *LNTS-RAW, comparative results regarding AAE, ARE*

CART-LAD and CART-LS provided the best and the worst underlying prediction regarding AAE and ARE values.

The models based on these two techniques alternatively have closer ranking to the perfect ranking along both ranges *I* and *II* as illustrated in Figure 4.108.

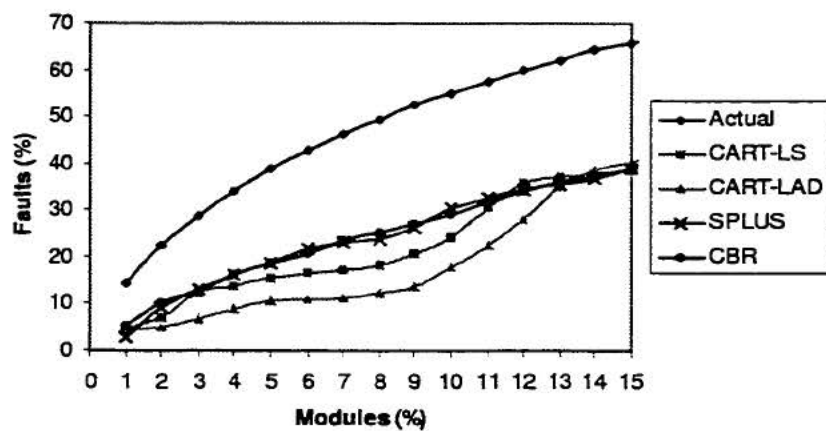


Figure 4.106: Close view of Alberg diagram for LNTS-RAW: CART-LS, CART-LAD, SPLUS and CBR.

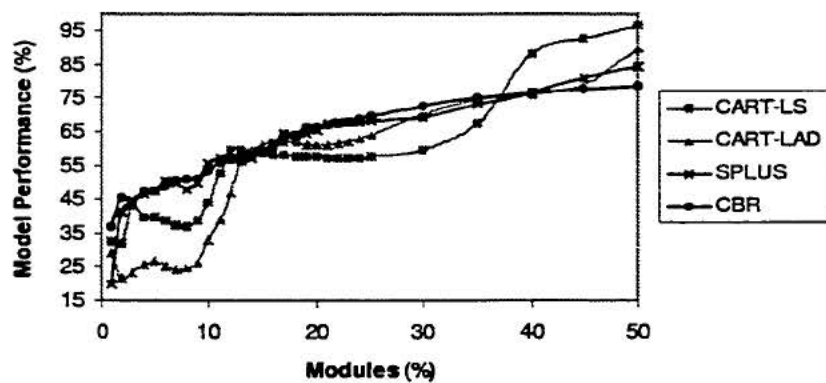


Figure 4.107: Performance of LNTS-RAW: CART-LS, CART-LAD, SPLUS and CBR.

For the most critical modules, CART-LS presents the predicted ranking closer to the perfect ranking than CART-LAD over the critical range as shown in Figure 4.109.

When focusing on performance, both techniques present unstable performance along the considered ranges. Considering range *I*, a ranking model based on CART-LS has higher performance for the former half of the range, while CART-LAD performs better than CART-LS for the later half of the range. However, the former half of range *I* would be more interesting than the later half because more faults are contained in the modules of the former half.

For range *II*, CART-LAD performs apparently better than CART-LS for the first half of the range. However, CART-LS visibly performs better than CART-LAD for the second half of range *II*.

Concisely, CART-LAD may have better prediction accuracy than CART-LS, but this doesn't mean that CART-LAD would perform better than CART-LS over all the cutoff ranges.

4.5.2 Experiment on LNTS-PCA

- **Comparative results of the underlying quantitative models, LNTS-PCA**

After conducting the principal components analysis, LNTS-PCA was obtained from LNTS-RAW data set. After the models were built using the fit data

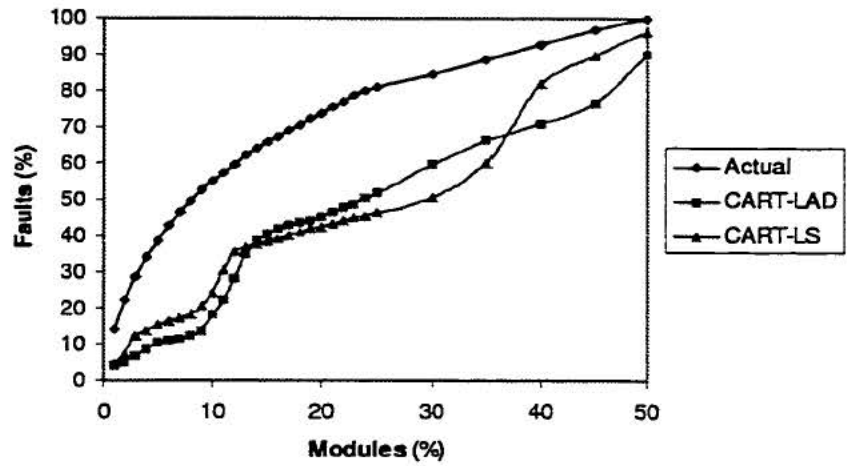


Figure 4.108: Alberg diagram for LNTS-RAW: CART-LS, CART-LAD

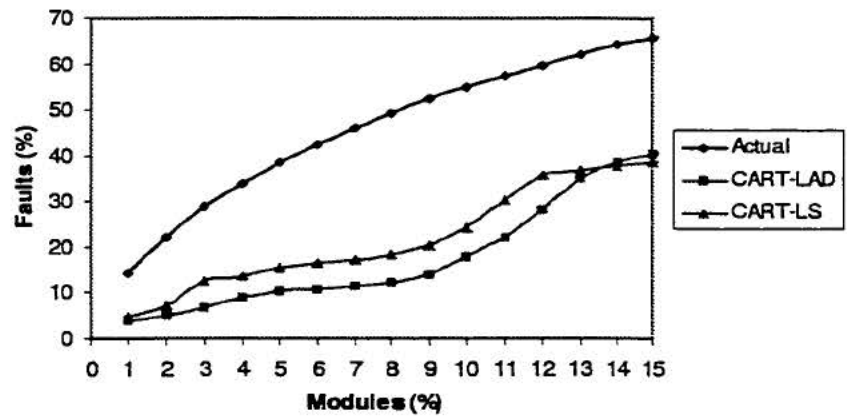


Figure 4.109: Close view of Alberg diagram for LNTS-RAW: CART-LS, CART-LAD

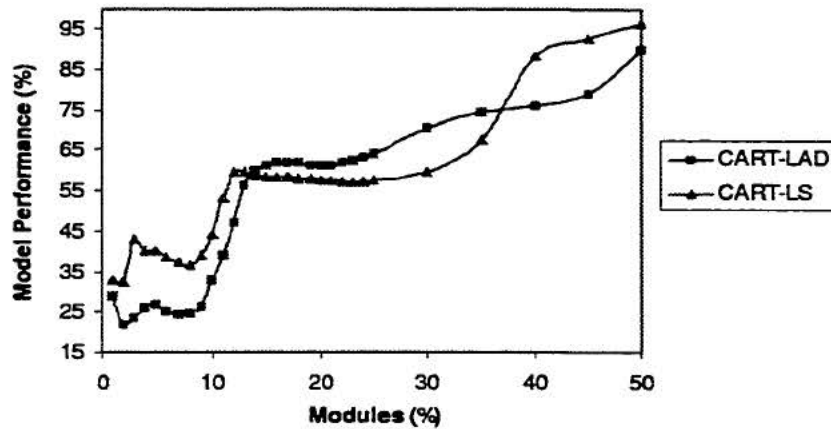


Figure 4.110: Performance of LNTS-RAW: CART-LS, CART-LAD

Table 4.13: LNTS-PCA, Comparative accuracy of underlying quantitative models

Model	Test data	
	<i>AAE</i>	<i>ARE</i>
CBR	1.231	0.624
ANN	1.291	0.682
MLR	1.254	0.671
CART-LS	1.304	0.694
CART-LAD	1.160	0.424
SPLUS	1.276	0.671

set, the test data set was applied. The results are shown in Table 4.13.

For case-based reasoning, the city-block distance and distance weighted average provided the best result among all available similarity functions and solution algorithms for LNTS-PCA [29].

Stepwise model selection (5% significance level) with LNTS-PCA data selected 3 out of 4 independent variables applied to Multiple Linear Regression. The

following model was obtained [29].

$$\begin{aligned} faults = & 1.24178336 + 0.75727622 \cdot DOMAIN1 + 0.058507056 \cdot DOMAIN2 \\ & + 0.34032970 \cdot DOMAIN3 \end{aligned}$$

When using CART-LS, the tree has 5 terminal nodes and uses 2 out of 4 independent variables. The tree built using CART-LAD has 4 leaf nodes and uses 3 out of 4 independent variables. For SPLUS, the preferred tree has 14 terminal nodes and uses 3 out of 4 independent variables [27].

CBR gave better prediction accuracy than MLR and ANN within group *I*. For group *II*, CART-LAD provided better prediction than CART-LS and SPLUS. Comparing all the underlying quantitative models, the order from the best to the worst accuracy is CART-LAD, CBR, MLR, SPLUS, ANN and CART-LS.

- **Comparative results of module-order models, LNTS-PCA**

1. *LNTS-PCA, Comparative Results for Group I*

The results of module-order models based on group *I* are illustrated in Figure 4.111, 4.112 and 4.113. The Alberg diagram and the close view of the most critical modules show that the three methods give very close predicted ranking models along both ranges *I* and *II*. When analyzing performances, the three techniques in group *I* also perform really close over all the considered ranges.

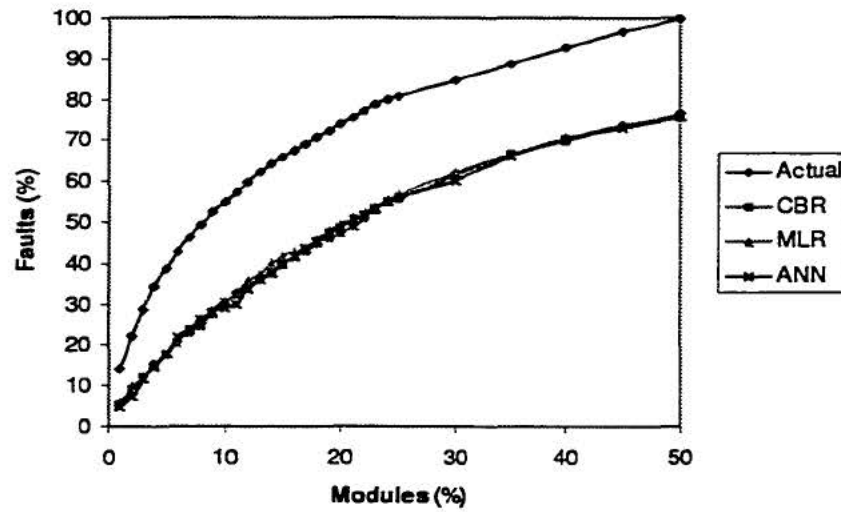


Figure 4.111: Alberg diagram for LNTS-PCA: CBR, MLR, ANN

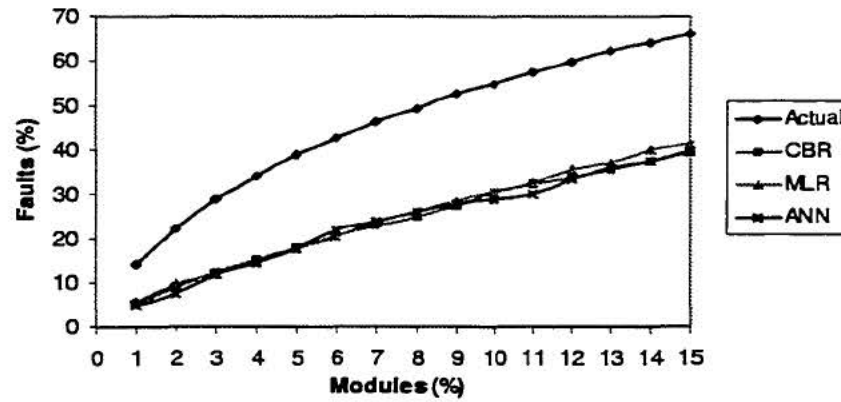


Figure 4.112: Close view of Alberg diagram for LNTS-PCA: CBR, MLR, ANN

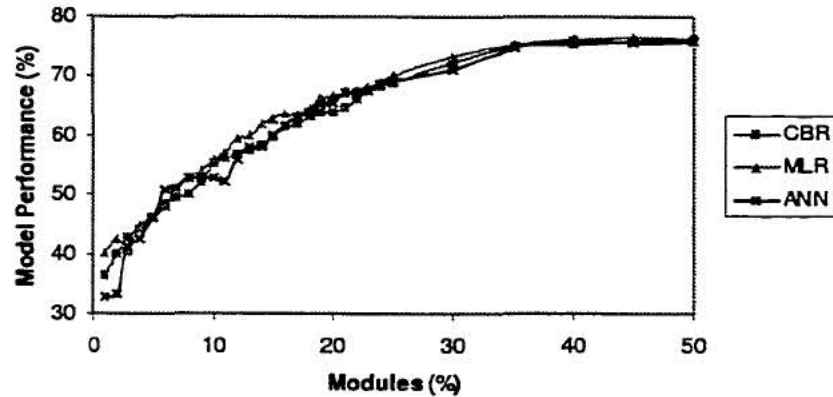


Figure 4.113: Performance of LNTS-PCA: CBR, MLR, ANN

2. *LNTS-PCA, Comparative Results for Group II*

The Alberg diagram in Figure 4.114 denotes that SPLUS seems to present closer predicted ranking to the perfect ranking than the two CART techniques over range *I*. However, ranking models using CART-LS and CART-LAD are closer to a perfect model than a model using SPLUS for some intervals of range *II*.

For the most critical modules, SPLUS presents the nearest ranking to the perfect ranking, and CART-LAD gives the farthest ranking from the perfect ranking as illustrated in Figure 4.115.

When analyzing performances for range *I*, SPLUS performs better than the other two methods. CART-LS presents lower performance than SPLUS. However, CART-LAD has considerably lower performance than both SPLUS and CART-LS for this range.

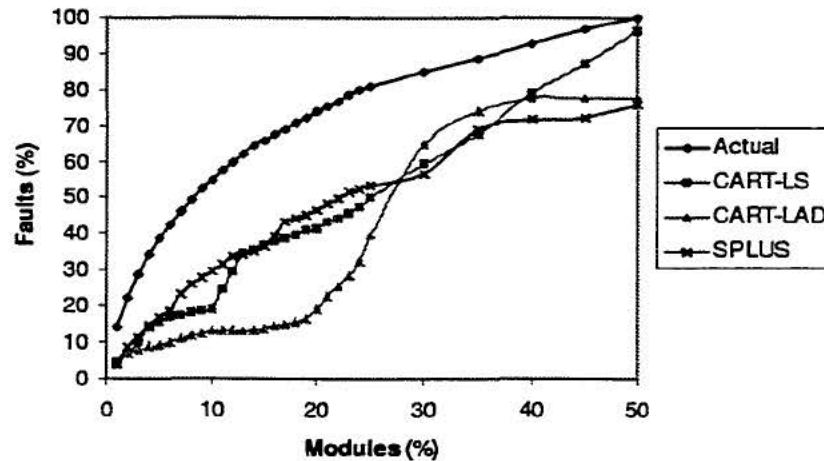


Figure 4.114: Alberg diagram for LNTS-PCA: CART-LS, CART-LAD, SPLUS

When considering performances over range *II*, both CART techniques perform better than SPLUS over range *II*. CART-LS has an increasing trend of performance, while CART-LAD presents varying trend of performance along this range. The performance diagram is shown in Figure 4.116.

CART-LAD has the best accuracy prediction within group *II*, but it presents considerably lower module-order modeling performance than others for the high interesting range, range *I*. This is one of several evidences that better prediction does not always yield better performance when module-order modeling.

3. LNTS-PCA, Group I and Group II Models Comparison

CBR is selected to represent all group *I*'s methods. The comparative results are illustrated in Figure 4.117, 4.118 and 4.119. We can easily notice again

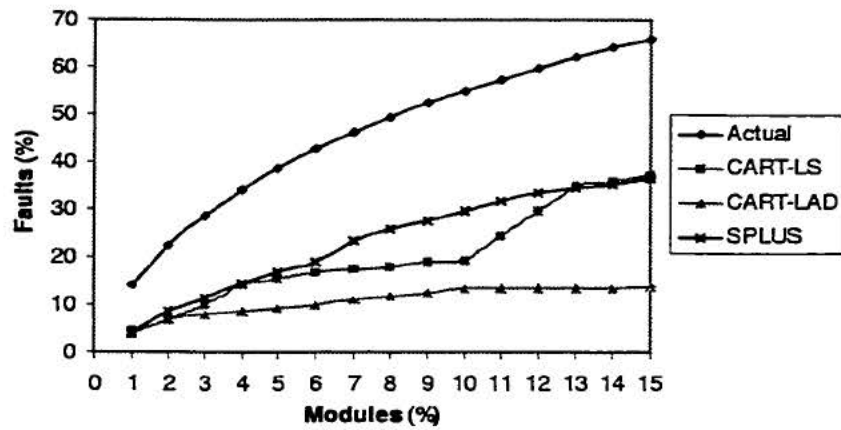


Figure 4.115: Close view of Alberg diagram for LNTS-PCA: CART-LS, CART-LAD, SPLUS

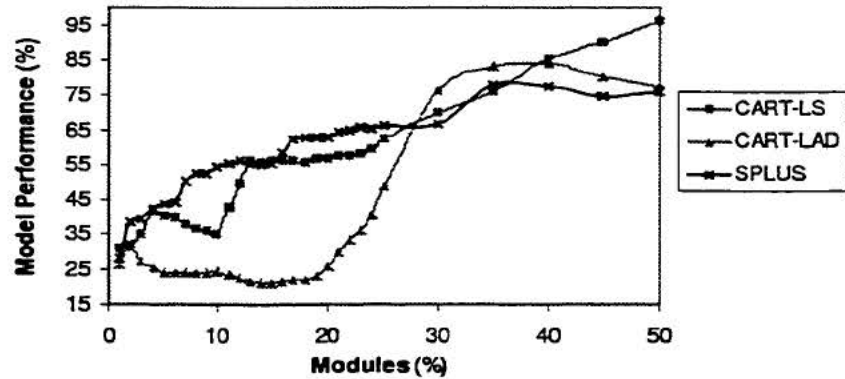


Figure 4.116: Performance of LNTS-PCA: CART-LS, CART-LAD, SPLUS

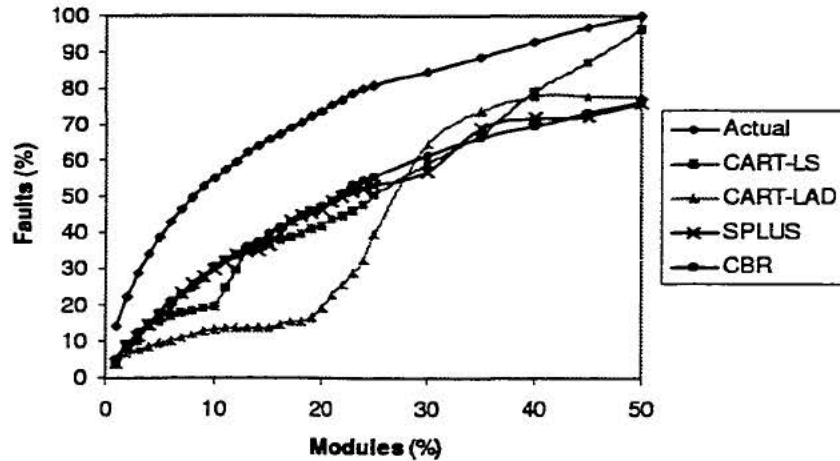


Figure 4.117: Alberg diagram for LNTS-PCA: CART-LS, CART-LAD, SPLUS and CBR

that CBR performs close to SPLUS. This yields that SPLUS also performs close to the other models in group *I*.

4. *LNTS-PCA, comparative results regarding AAE and ARE*

CART-LAD and CART-LS provided the best and the worst prediction. However, CART-LAD doesn't perform better than CART-LS. When focusing on range *I*, CART-LS performs considerably better than CART-LAD. We can easily notice the large gap existing between the two curves. For range *II*, CART-LAD presents better performance for some intervals of the range, but at the end of range *II*, it doesn't perform as well as CART-LS. For the most critical ranges, CART-LS obviously presents closer ranking to the perfect ranking than CART-LAD. The graphical presentation is shown in Figure 4.120,

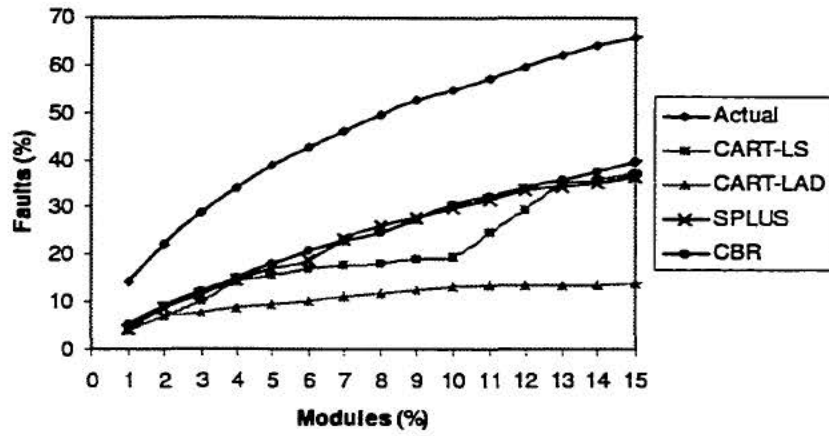


Figure 4.118: Close view of Alberg diagram for LNTS-PCA: CART-LS, CART-LAD, SPLUS and CBR

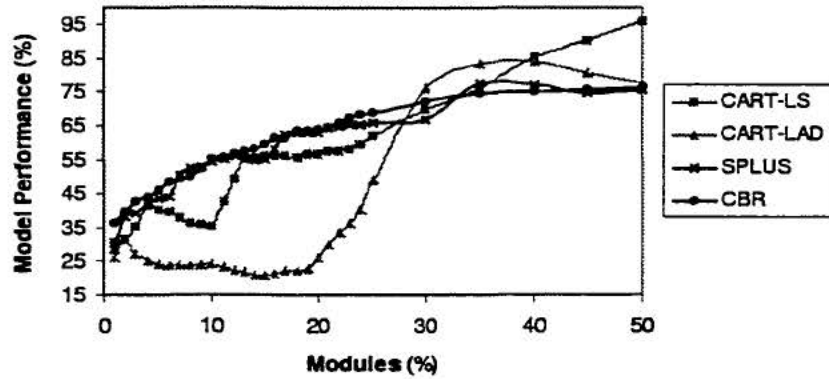


Figure 4.119: Performance of LNTS-PCA: CART-LS, CART-LAD, SPLUS and CBR

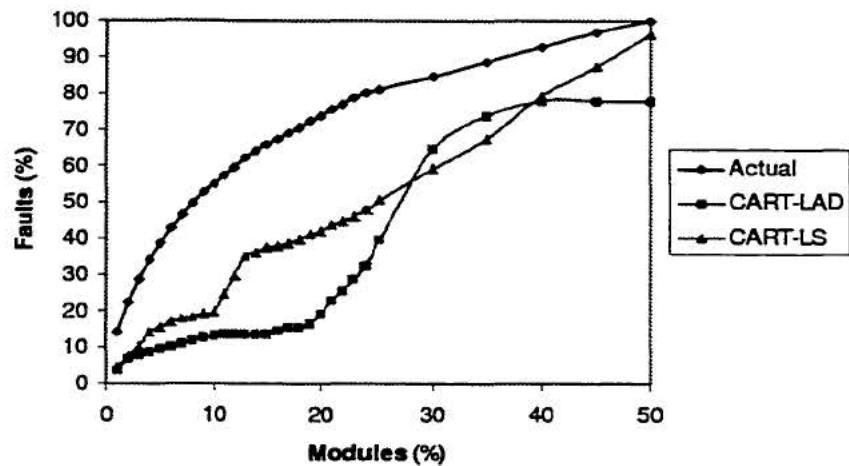


Figure 4.120: Alberg diagram for LNTS-PCA: CART-LS, CART-LAD

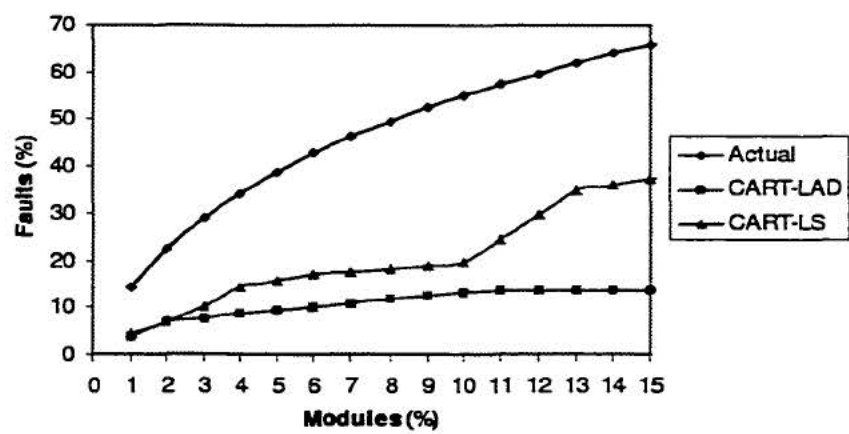


Figure 4.121: Close view of Alberg diagram for LNTS-PCA: CART-LS, CART-LAD

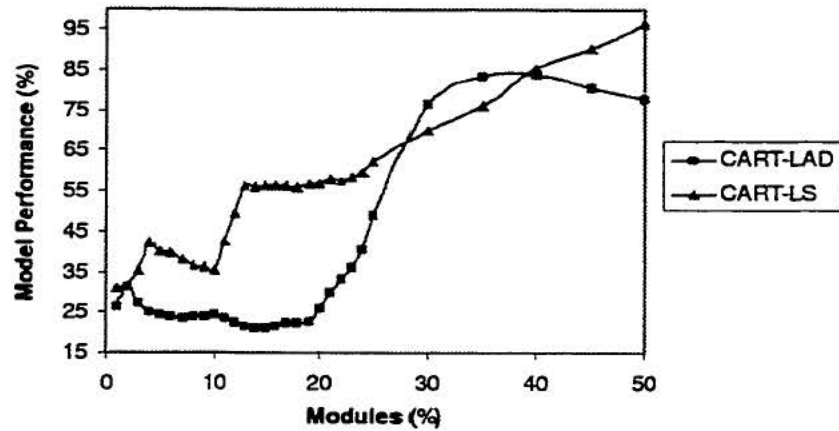


Figure 4.122: Performance of LNTS-PCA: CART-LS, CART-LAD

4.121 and 4.122.

4.6 Comparing module-order models based on RAW and PCA metrics

Prior research stated that the use of principal components analysis does not improve the prediction accuracy for tree-modeling [27]. For non tree-modeling techniques (CBR, MLR, ANN), when comparing the prediction accuracy using PCA and RAW metrics [29], PCA gave better accuracy than RAW for the LLTS case study. However, RAW presented better accuracy than PCA for NT and LNTS. Therefore, we can conclude that PCA did not improve the prediction accuracy for tree models and didn't systematically improve the prediction accuracy for the other models.

For our research, we investigate the benefits of using principal components analysis in module-order modeling. We will compare the module-order models built using the RAW metrics and their principal components.

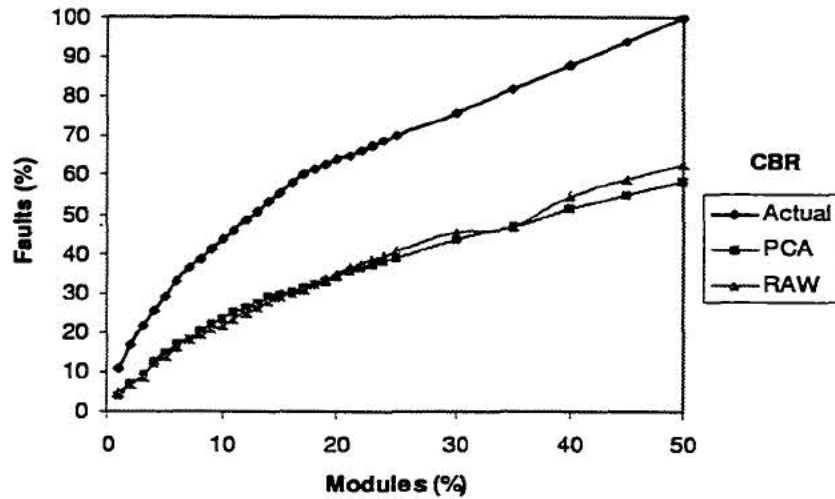


Figure 4.123: Alberg diagram for LLTS PCA and RAW comparison release 4: CBR

4.6.1 Comparing module-order models for LLTS

1. *Comparative Results for Group I*

For LLTS data, the module-order models built using RAW metrics are very close to the ones built using PCA metrics for all methods in group *I*. We can see the obvious examples in Figure 4.123, 4.124 and 4.125.

2. *Comparative Results for Group II*

When considering CART-LS, the module-order models built using RAW metrics give closer ranking to the perfect ranking than the ones using PCA metrics for all releases. We show an example of the comparison in Figure 4.126

For CART-LAD, the results are similar to CART-LS. The ordering models built using RAW metrics presents closer ranking to the perfect ranking than

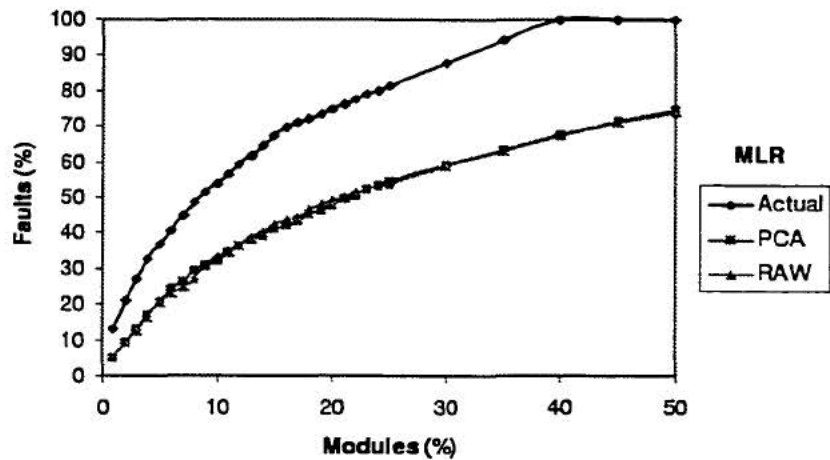


Figure 4.124: Alberg diagram for LLTS PCA and RAW comparison release 2: MLR

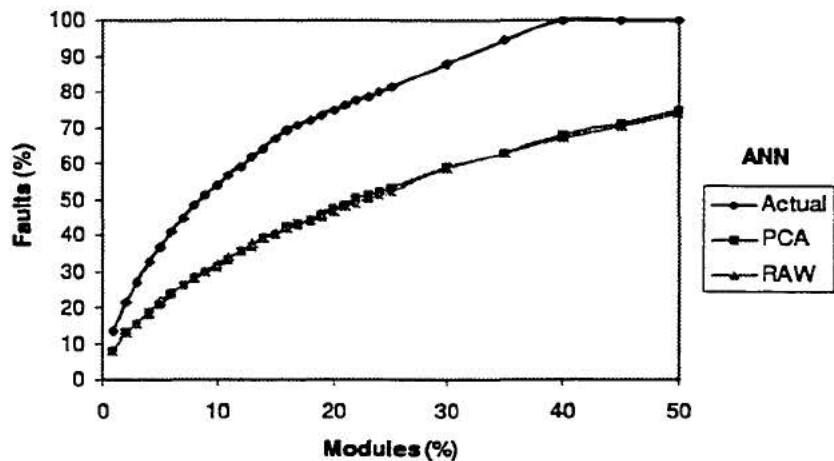


Figure 4.125: Alberg diagram for LLTS PCA and RAW comparison release 2: ANN

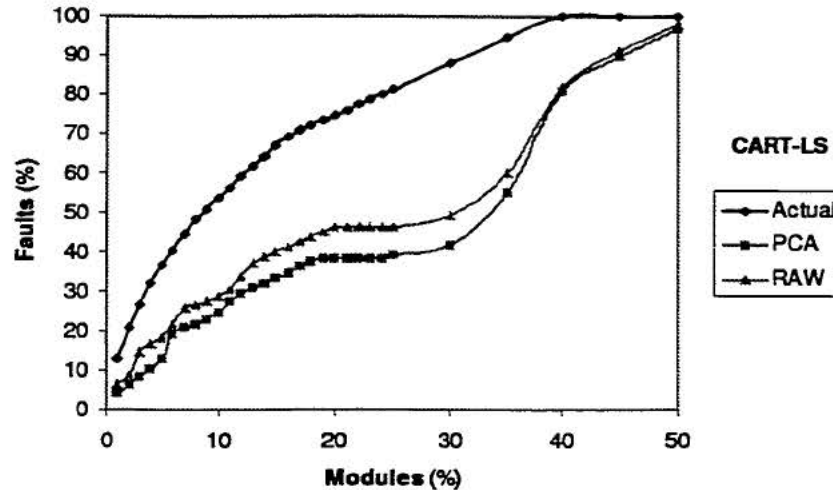


Figure 4.126: Alberg diagram for LLTS PCA and RAW comparison release 2: CART-LS

the ones using PCA metrics for release 2 and 4. For release 3, the two models alternatively have closer ranking to the perfect ranking, but the one built using RAW metrics present closer ranking for the most critical modules. The example of release 2 and 3 are shown in Figure 4.127 and 4.128.

SPLUS presents similar results to the group *I* techniques. The module-order models built using RAW metrics are very close to the ones built using PCA metrics as shown in Figure 4.129.

Concisely, It was observed that when comparing PCA and RAW for the three group *I*'s methods, the module-order models are very close. For group *II*, RAW metrics give better models than PCA for the two CART techniques. No improvement was observed for SPLUS. Therefore, the use of principal components

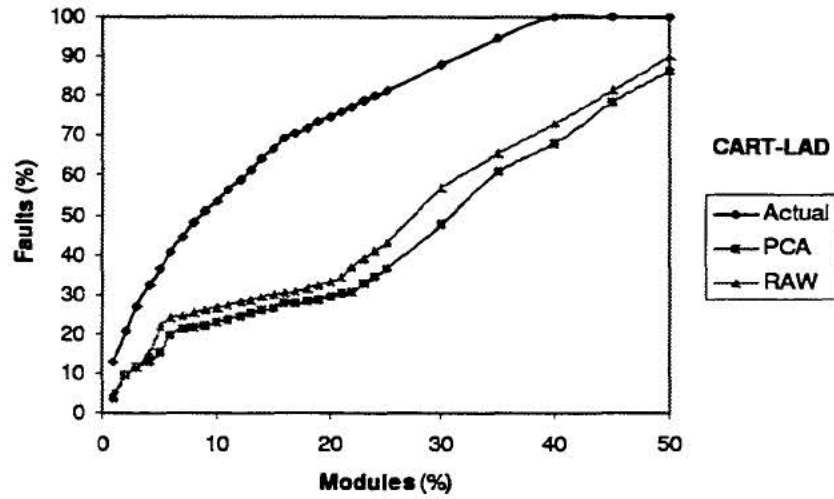


Figure 4.127: Alberg diagram for LLTS PCA and RAW comparison release 2: CART-LAD

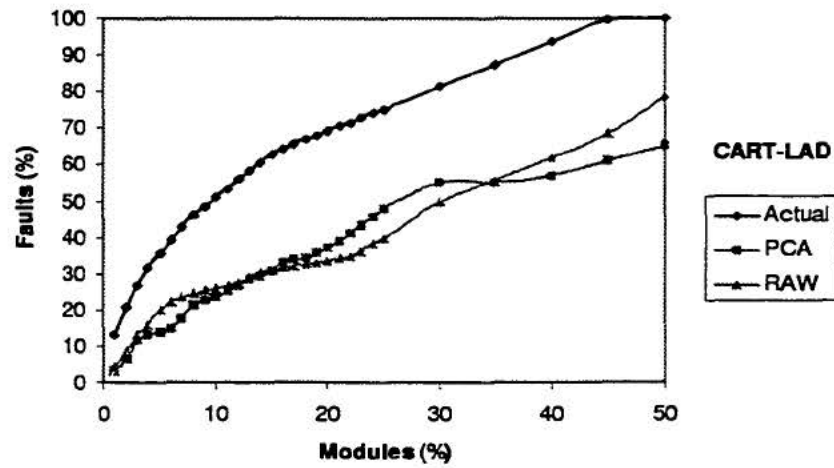


Figure 4.128: Alberg diagram for LLTS PCA and RAW comparison release 3: CART-LAD

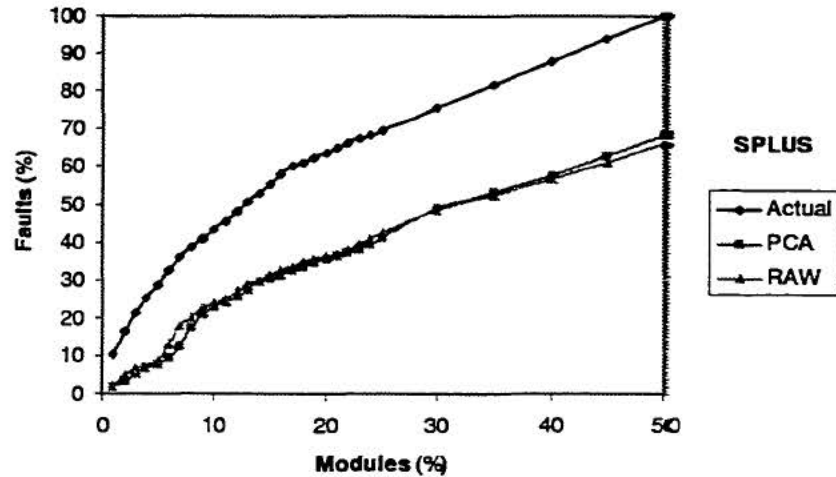


Figure 4.129: Alberg diagram for LLTS PCA and RAW comparison release 4: SPLUS

analysis does not yield any improvement when module-order modeling for the LLTS case study.

4.6.2 Comparing module-order models of NT

1. Comparative Results for Group I

The module-order models of group *I* built using RAW and PCA metrics are very close to each other. When using PCA metrics for CBR, the model is very close to the one using RAW metrics as illustrated in Figure 4.130.

For MLR, the predicted ranking using PCA is closer to the perfect ranking than the one using RAW over range *I*, but the difference is not large, and both predicted rankings become close to each other over range *II* as shown in Figure 4.131.

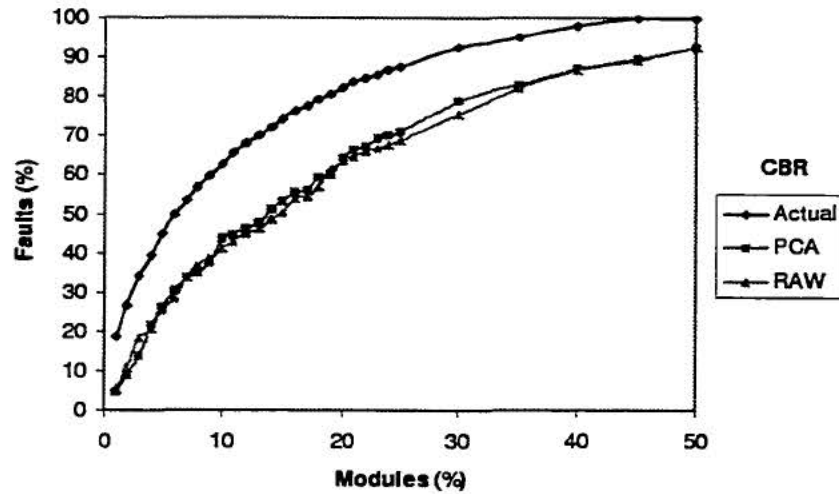


Figure 4.130: Alberg diagram for NT PCA and RAW comparison: CBR

For ANN, the model built using RAW is slightly closer to the the perfect ranking than the one using PCA metrics as shown in Figure 4.132.

2. Comparative Results for Group II

The module-order model using PCA is visibly better than the one using RAW metrics for CART-LS. We obviously see the considerable difference between PCA and RAW over range *I*. Figure 4.133 shows the comparative results for CART-LS.

For CART-LAD, the model built using RAW is better than the one using PCA metrics over the considered ranges. The two models are very close to each other for the beginning of the cutoff range *I*. Afterwards, the model using RAW presents a closer ranking to the perfect ranking than the one using PCA

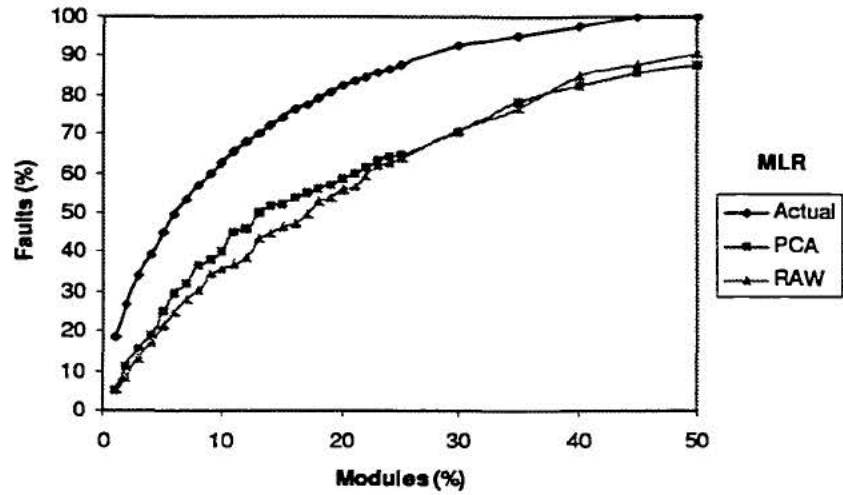


Figure 4.131: Alberg diagram for NT PCA and RAW comparison: MLR

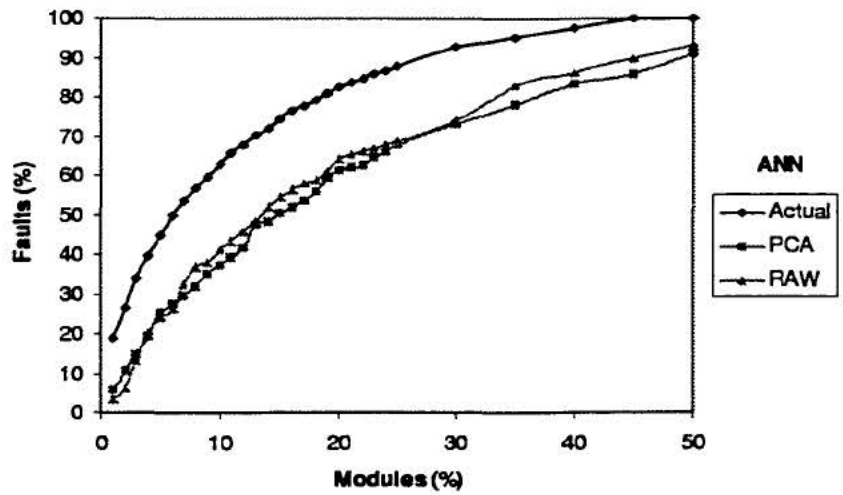


Figure 4.132: Alberg diagram for NT PCA and RAW comparison: ANN

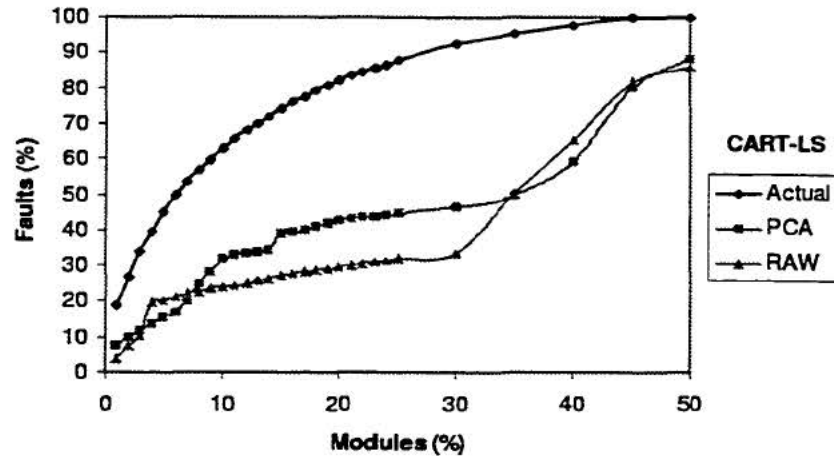


Figure 4.133: Alberg diagram for NT PCA and RAW comparison: CART-LS as shown in Figure 4.134. The same result is also seen for SPLUS as illustrated in Figure 4.135.

To summarize, using PCA metrics improves the module-order models for MLR and CART-LS. However, for the other techniques in group *I*, the models using PCA are very close to the ones using RAW metrics. For the other methods in group *II*, using RAW metrics yielded better results than using PCA when module-order modeling.

4.6.3 Comparing module-order models for LNTS

1. Comparative Results for Group *I*

The module-order models using RAW metrics are very close to the ones using PCA metrics for all group *I*'s methods as shown in Figure 4.136, 4.137 and 4.138.

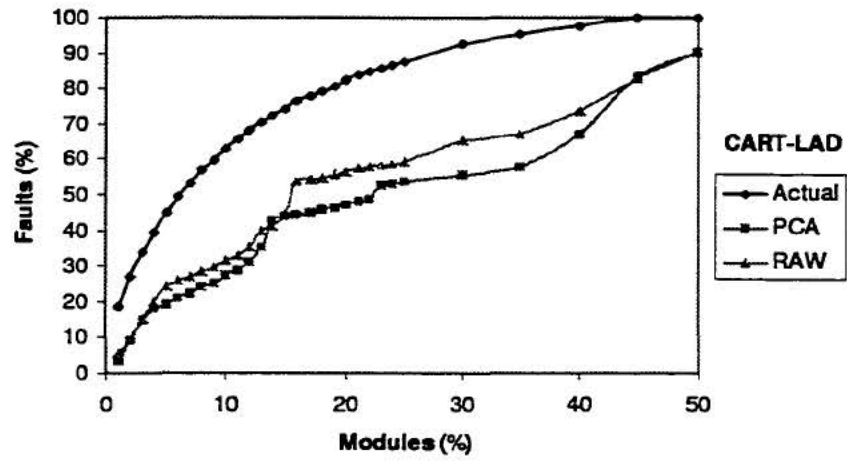


Figure 4.134: Alberg diagram for NT PCA and RAW comparison: CART-LAD

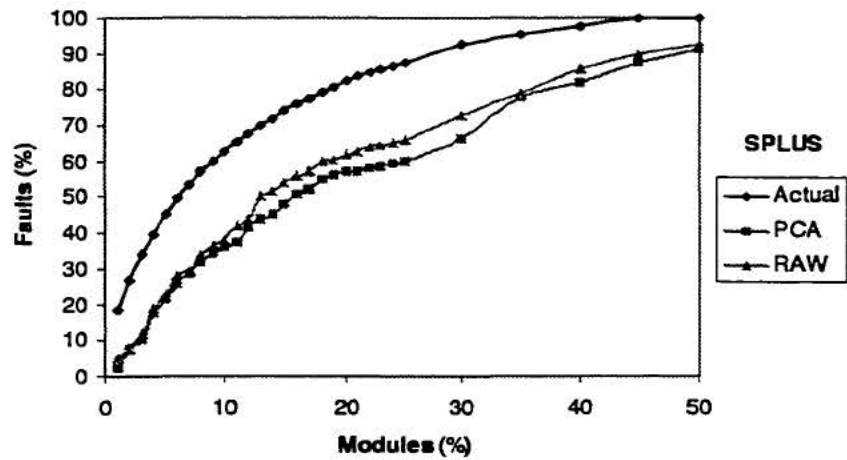


Figure 4.135: Alberg diagram for NT PCA and RAW comparison: SPLUS

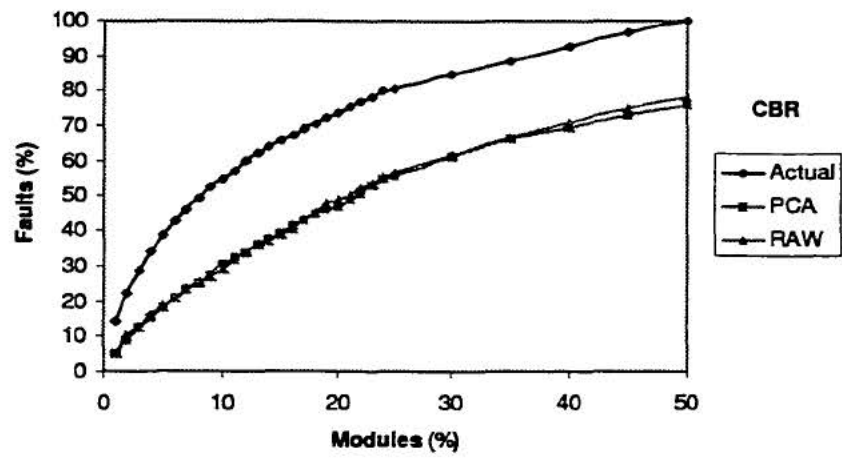


Figure 4.136: Alberg diagram for LNTS PCA and RAW comparison: CBR

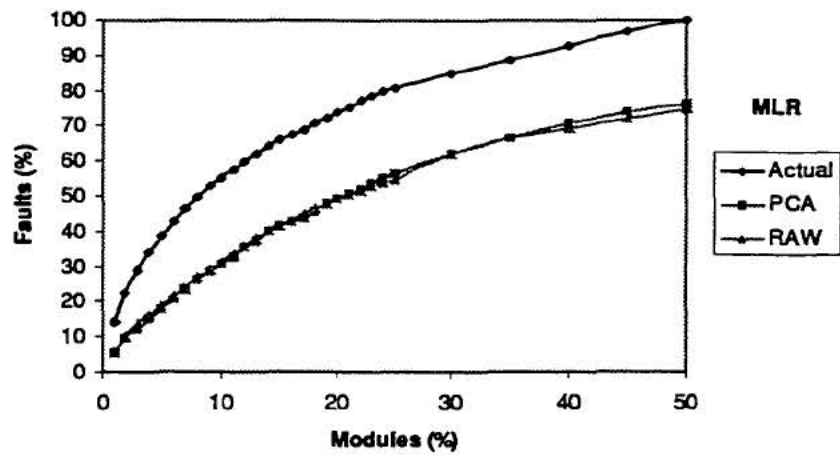


Figure 4.137: Alberg diagram for LNTS PCA and RAW comparison: MLR

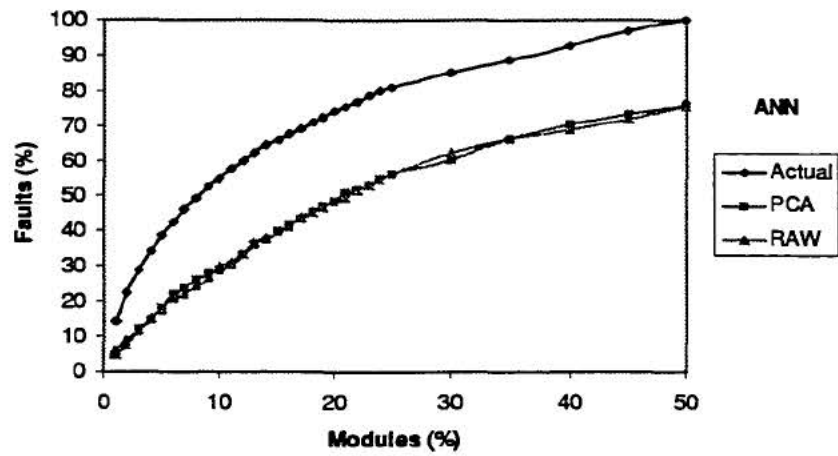


Figure 4.138: Alberg diagram for LNTS PCA and RAW comparison: ANN

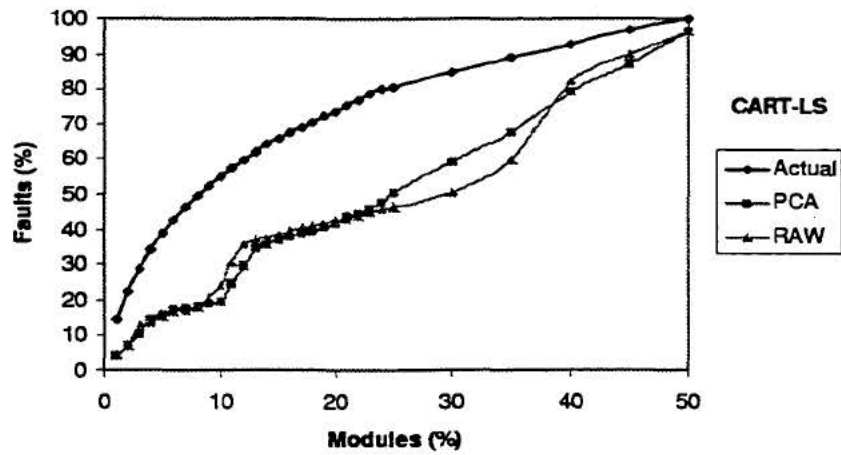


Figure 4.139: Alberg diagram for LNTS PCA and RAW comparison: CART-LS

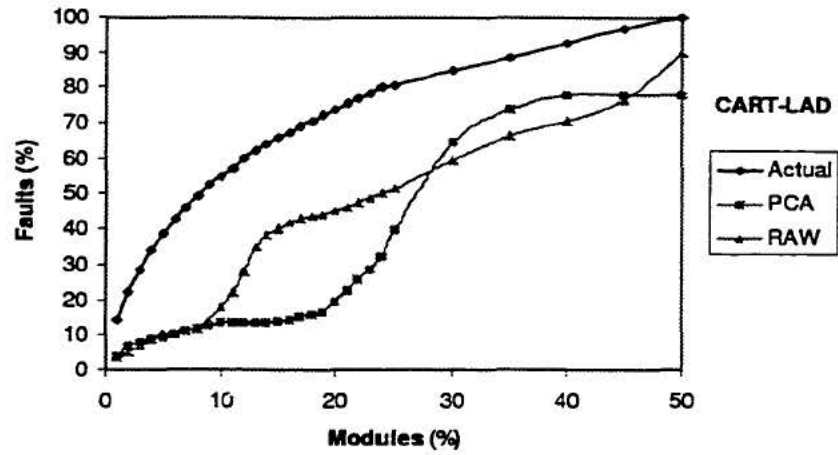


Figure 4.140: Alberg diagram for LNTS PCA and RAW comparison: CART-LAD

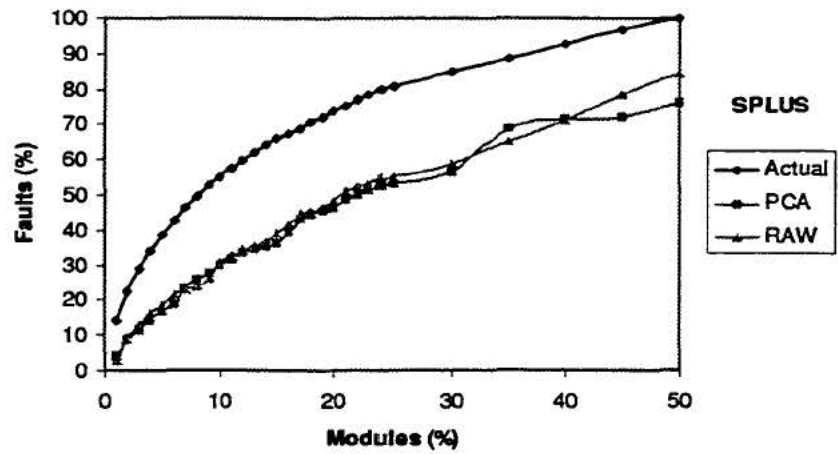


Figure 4.141: Alberg diagram for LNTS PCA and RAW comparison: SPLUS

2. comparative Results for Group II

When comparing PCA with RAW for CART-LS, the two models are very close for range *I*, but they alternatively present closer ranking to the perfect ranking for range *II* as shown in Figure 4.139.

For CART-LAD, the model using RAW is considerably better than the one using PCA metrics over range *I*, but they alternatively present closer ranking to the perfect ranking over range *II* as shown in Figure 4.140.

For SPLUS, the model using PCA is very close models to one using RAW metrics over the considered ranges as shown in Figure 4.141

To summarize, the module-order models using PCA are very close to ones using RAW metrics for three group *I*'s methods and SPLUS. For the two CART methods, the models using PCA are not better than ones using RAW metrics for the range of higher interest. Therefore, the use of principal components analysis does not yield any benefit when module-order modeling for the LNTS case study.

Chapter 5

CONCLUSIONS

In this chapter, we present our conclusions based on the results of experiments on module-order modeling.

5.1 Overview

A module-order model predict the rank-order of modules based on a quantitative quality factor. In this research, we used number of faults as a quality factor. An empirical study based on various underlying quantitative software quality prediction methods was performed. Those underlying quantitative prediction models are Case-base Reasoning (CBR), Multiple Linear Regression (MLR), Artificial Intelligent Networks (ANN), CART Least Square (CART-LS), CART Least Absolute Deviation (CART-LAD) and SPLUS algorithms.

Three case studies of full-scale industrial software systems were used to compare the module-order modeling performance of the different underlying techniques.

Both original RAW data and preprocessed PCA data set were applied to the experiments. We will conclude with the lessons learned from this study following the three objectives discussed in the introduction of this thesis.

The first objective is to study the behavior of module-order models based on five different underlying techniques. The conclusion can be grouped as the following.

1. When considering CBR, MLR, and ANN the performances remained very close to each other when module-order modeling.
2. For tree-modeling group, CART-LS, CART-LAD and SPLUS, the tree techniques present different module-order models for all three case studies. The models based on the two CART methods present varying behaviors and performances along the considered range. SPLUS provided better performances than the CART methods. In addition SPLUS provides the most robust model due to the least variation of $\phi(c)$ in this group. The diagram presented the performance from these tree-modeling techniques have different trend of path.
3. When comparing non tree-modeling with tree-modeling group, CBR, MLR and ANN perform close to SPLUS when module-order modeling compared to CART techniques.
4. We can not conclude which underlying technique has the best performance when module-order modeling. This depends on the particular cutoff percentile the manager wants to select. For example in the experiments, CART-LS

performs considerably better than other underlying techniques around the 40-50 percentile for all case studies, however it frequently provides poorer performances than other techniques around the beginning of the considered range.

For the second objective, we investigated the benefits of using principal components analysis when module-order modeling. When considering the non tree-modeling group, the models built using PCA remained very close to the ones built using RAW. For the tree-modeling group, RAW metrics usually yielded better models than PCA metrics for all three case studies. This leads us to the conclusion that the use of PCA doesn't yield better results when module-order modeling.

For the third objective, we have verified that better prediction accuracy, doesn't always yield better performance when module-order modeling. We can see several evidences according to the comparative results. Prior research [27] stated that CART-LAD had better prediction accuracy than other techniques. However, CART-LAD did not perform better than the other techniques when module-order modeling for all three case studies.

Overall, any underlying quantitative method can be applied to module-order modeling. The performance of a rank-order models may be differ from one algorithm to another and from one case study to another. In addition, AAE and ARE value seem not to be good indicators to define a good prediction model for module-order modeling.

5.2 Future Work

Module-order modeling can be further investigated by using different underlying quantitative techniques and applying it to a wide variety of systems other than a telecommunications system.

BIBLIOGRAPHY

- [1] B. Beizer. *Software Testing Techniques*. Van Nostand Reinhold, New York, 2d edition, 1990.
- [2] Y. Berkovich. Software quality prediction using case-based reasoning. Master's thesis, Florida Atlantic University, Boca Raton, FL USA, Aug. 2000. Advised by Taghi M. Khoshgoftaar.
- [3] L. C. Briand, V. R. Basili, and C. J. Hetmanski. Developing interpretable models with optimized set reduction for identifying high-risk software components. *IEEE Transactions on Software Engineering*, 19(11):1028–1044, 1993.
- [4] W. R. Dillon and M. Goldstein. *Multivariate Analysis: Methods and Applications*. John Wiley & Sons, New York, 1984.
- [5] N. E. Fenton and S. L. Pfleeger. *Software Metrics*. PWS Publishing Company, New York, 2d edition, 1997.
- [6] J. P. Hudepohl, S. J. Aud, T. M. Khoshgoftaar, E. B. Allen, and J. Mayrand. EMERALD: Software metrics and models on the desktop. *IEEE Software*, 13(5):56–60, Sept. 1996.
- [7] J. P. Hudepohl, S. J. Aud, T. M. Khoshgoftaar, E. B. Allen, and J. Mayrand. EMERALD: Software metrics and models on the desktop. In *Proceedings of the Fourth International Symposium on Assessment of Software Tools*, pages 111–112, Toronto, May 1996. IEEE Computer Society. Extended abstract of [6].
- [8] W. D. Jones, J. P. Hudepohl, T. M. Khoshgoftaar, and E. B. Allen. Application of a usage profile in software quality models. In *Proceedings of*

the Third European Conference on Software Maintenance and Reengineering, pages 148–157, Amsterdam, Netherlands, Mar. 1999. IEEE Computer Society.

- [9] T. M. Khoshgoftaar and E. B. Allen. A practical classification rule for software quality models. Technical Report TR-CSE-97-56, Florida Atlantic University, Boca Raton, Florida USA, Nov. 1997.
- [10] T. M. Khoshgoftaar and E. B. Allen. Ordering fault-prone software modules. Technical Report TR-CSE-98-9, Florida Atlantic University, Boca Raton, Florida USA, Feb. 1998.
- [11] T. M. Khoshgoftaar and E. B. Allen. Predicting the order of fault-prone modules in legacy software. In *Proceeding of the Ninth International Symposium on Software Reliability Engineering*, pages 344–353, Paderborn, Germany, Nov. 1998. IEEE Computer Society.
- [12] T. M. Khoshgoftaar and E. B. Allen. A comparative study of ordering and classification of fault-prone software modules. *Empirical Software Engineering*, 4:159–186, 1999.
- [13] T. M. Khoshgoftaar and E. B. Allen. Software quality modeling: The software measurement analysis and reliability toolkit. In *Proceeding of the Twelfth IEEE International Conference on Tools with Artificial Intelligence*, pages 54–61. IEEE Computer Society, Nov. 2000.
- [14] T. M. Khoshgoftaar, E. B. Allen, N. Goel, A. Nandi, and J. McMullan. Detection of software modules with high debug code churn in a very large legacy system. In *Proceedings of the Seventh International Symposium on Software Reliability Engineering*, pages 364–371, White Plains, NY, Oct. 1996. IEEE Computer Society.
- [15] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl. Accuracy of software quality models over multiple releases. *Annals of Software Engineering*, 9:103–116, 2000.
- [16] T. M. Khoshgoftaar, E. B. Allen, K. S. Kalaichelvan, and N. Goel. Early quality prediction: A case study in telecommunications. *IEEE Software*, 13(1):65–71, Jan. 1996.

- [17] T. M. Khoshgoftaar, J. C. Munson, B. B. Bhattacharya, and G. D. Richardson. Predictive modeling techniques of software quality from software measures. *IEEE Transactions on Software Engineering*, 18(11):979–987, Nov. 1992.
- [18] K. E. E. L. C. Briand and S. Morasca. On the application of measurement theory in software engineering. *Empirical Software Engineering: An International Journal*, 1(1):61–88, 1996.
- [19] Y. LeCun. A learning procedure for asymmetric network. *Cognitiva*, 85:599–604, 1985.
- [20] C. T. Lin and C. S. G. Lee. *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems*. Prentice Hall, Inc., Upper Saddle River, New Jersey, 1996.
- [21] J. Maryrand and F. Coallier. System acquisition based on software product assessment. In *Proceeding of the 18th International Conference on Software Engineering*, pages 210–219, Berlin, Germany, Mar. 1996. IEEE Computer Society.
- [22] J. C. Munson and T. M. Khoshgoftaar. The detection of fault-prone programs. *IEEE Transactions on Software Engineering*, 18(5):423–433, 1992.
- [23] R. H. Myers. *Classical and Modern Regression with Applications*. PWS-KENT Publishing Company, Boston, 1990. Duxbury Series.
- [24] N. Ohlsson and H. Alberg. Predicting fault-prone software modules in telephone switches. *IEEE Transaction on Software Engineering*, 22(12):886–894, 1996.
- [25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Parallel Distributed Processing*, volume 1, chapter 8. MIT Press, Cambridge, MA, 1986.
- [26] N. F. Schneidewind. Methodology for validating software metrics. *IEEE Transaction on Software Engineering*, 18(5):410–422, May 1992.

- [27] N. Seliya. Software fault prediction using tree based modeling. Master's thesis, Florida Atlantic University, Boca Raton, FL USA, Aug. 2001. Advised by Taghi M. Khoshgoftaar.

- [28] R. Shan. Modeling software quality with classification trees using principal components analysis. Master's thesis, Florida Atlantic University, Boca Raton, FL USA, Dec. 1999. Advised by Taghi M. Khoshgoftaar.

- [29] N. Sundaresh. An empirical study of analogy based software fault prediction. Master's thesis, Florida Atlantic University, Boca Raton, FL USA, May 2001. Advised by Taghi M. Khoshgoftaar.

- [30] L. G. Votta and A. A. Porter. Experimental software engineering: A report on the state of the art. In *Proceeding of the Seventeenth International Conference on Software Engineering*, pages 277–279, Seattle, WA, Apr. 1995. IEEE Computer Society.

- [31] M. C. Yovitz, G. T. Jacobi, and G. Goldstein. *Self Organizing Systems*. Spartan Books, Washington, DC, 1962.

