INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

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

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

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

**An Automated, Incremental
Resource Function
Construction Technique**

**by
Marc Courtois**

A thesis submitted to the Faculty of Graduate Studies in partial fulfillment of
the requirements for the degree of

Master of Science (ISS)

Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering

Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada, K1S 5B6

August, 1997

© Copyright 1997, Marc Courtois

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*

Our file *Notre référence*

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-27044-0

The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

**An Automated, Incremental
Resource Function
Construction Technique**

submitted by Marc Courtois, B.Com.,
in partial fulfillment of the requirements for
the degree of Master of Science (ISS)

Chair, Department of Systems and Computer Engineering

Thesis Supervisor

Carleton University
August, 1997

Abstract

A resource function describes the cost of executing a software component and is used to provide estimates of parameter values within a software performance model. A technique is described for constructing resource functions to within a required accuracy. The accuracy requirement of a software performance model is used as a guideline to set the accuracy requirement of the resource function. A non-parametric, regression tool, MARS, is employed to build resource functions from measurement data, and an accuracy estimate, based on pseudo-confidence intervals, has been developed to gauge the MARS model's accuracy. The technique incrementally takes measurements, models the data with MARS and estimates the accuracy of the resulting resource function. This process is continued until the MARS model satisfies the required accuracy. A point selection algorithm has also been created to determine the best point at which to take the next measurement, should that be necessary. The technique has been implemented as a Tcl script and integrated with the Performance Workbench. It has been successfully validated against a variety of known functions, with linear, non-linear and even discontinuous surfaces, and it has been used to construct resource functions for the components of the ACE Event Server. This technique limits the amount of effort involved in building a resource function by automatically determining the experimental design, by employing non-parametric regression to model the data, and by only using as many measurements as are needed to satisfy the accuracy requirement.

Acknowledgment

I would like to thank Professor C.M. Woodside for his guidance throughout this thesis. It has been a privilege to study under him.

Table of Contents

Abstract	ii
Acknowledgment	iii
Table of Contents	iv
List of Figures	vii
List of Tables	ix
CHAPTER 1 Introduction	1
1.0 Motivation	1
1.1 An Illustration of a Resource Function	6
1.2 Contributions of this Thesis	9
1.3 Approach Taken in this Thesis	10
1.3.1 Type of detail considered when constructing a resource function	10
1.3.2 Fitting the data	10
1.3.3 Estimating the accuracy of a resource function	11
1.3.4 Setting the required accuracy of the resource function	11
1.4 An Incremental Procedure for Constructing Resource Functions	12
1.5 Overview of Chapters	13
CHAPTER 2 Resource Functions	15
2.0 Overview	15
2.1 Definitions	16
2.2 Layered Queuing	18
2.3 Impact of Resource Function Accuracy on the Performance Model	21
2.3.1 Notation	23
2.3.2 Quantifying the impact of resource function error on performance model accuracy	24
2.3.3 Determining the Elasticity of a Performance Model Response	24
2.4 Constructing Resource Functions	25
2.4.1 Principles of Experimental Design	25
2.4.2 Recent Work in Software Resource Demand Characterization	29

2.5 Summary	34
CHAPTER 3 MARS (Multivariate Adaptive Regression Splines)	36
3.0 Overview	36
3.1 Notation	37
3.2 On General Linear Regression	38
3.2.1 Errors, Cross Validation, and Generalized Cross Validation	40
3.2.2 Confidence Bands	42
3.2.3 A Problem with Regression Modeling	45
3.3 On Splines	46
3.3.1 Truncated Power Splines	48
3.3.2 Truncated Power Splines in Multiple Dimensions	51
3.4 On Regression Splines	52
3.5 Multivariate Adaptive Regression Splines - MARS	53
3.5.1 Degrees of Freedom and the Maximum Number of Basis Functions	57
CHAPTER 4 A Method for Limiting Detail	59
4.0 Overview	59
4.1 Choosing the Accuracy	60
4.2 Estimating the Accuracy of a MARS Model	62
4.2.1 GCV-Based Pseudo-Confidence Intervals	62
4.3 Improving the Accuracy of a MARS Model	66
4.4 A Strategy for Selecting Additional Points	71
4.5 An Incremental Resource Function Construction Technique	77
CHAPTER 5 Implementation and Validation	79
5.0 Overview	79
5.1 An Extension to the Performance Workbench	80
5.1.1 The Experiment Launcher	80
5.1.2 Smartexec	83
5.1.2.1 Format of the Input File	85
5.1.2.2 Format of the Output File	87
5.1.3 The Resource Function Manager	88

5.1.4 Choosing the Next Design Point	89
5.1.5 Complexity	90
5.2 Validation	91
5.2.1 Validating EA	91
5.2.1.1 Randomly Generating the Functions	93
5.2.1.2 Noise	93
5.2.1.3 In Conjunction with the Point Selection Algorithm	94
5.2.1.4 Goal	94
5.2.1.5 Results	95
5.2.2 Validating the Construction Process	97
5.2.2.1 A Linear Function	98
5.2.2.2 A Log Function	100
5.2.2.3 A 2nd Order Polynomial	103
5.2.2.4 A discontinuous function	106
5.3 Summary	110
CHAPTER 6 Case Study: The ACE Event Server	112
6.0 Overview	112
6.1 The ACE Event Server	113
6.1.1 The Components Involved	113
6.1.2 Putting the Components Together in the ACE Event Server	117
6.1.3 Call Structure and Threading of the ACE Event Server	119
6.1.4 A Performance Model for the ACE Event Server	122
6.2 The Experiments and Resource Functions	124
6.2.1 Supplier Handler	125
6.2.2 Consumer Router	128
6.2.3 Comparison with Simple Linear Regression	133
6.3 Impact of Execution Time Estimates on the Performance Model	135
CHAPTER 7 Conclusions	138
7.0 Success	138
7.1 Caveats and Future Work	140
CHAPTER 8 Bibliography	142

List of Figures

FIGURE 1. Pozzetti et al's Signature	7
FIGURE 2. Resource Function Submodel within Performance Model	22
FIGURE 3. Visualizing a 2K factorial design with three parameters	27
FIGURE 4. Model and Confidence Interval in the Log Domain	64
FIGURE 5. Increasing the accuracy of MARS, N=5 points	67
FIGURE 6. N=10 points	67
FIGURE 7. N=20 points	68
FIGURE 8. N=40 points	68
FIGURE 9. ARF of 1000 Randomly Chosen Points	76
FIGURE 10. Correctness of EA	96
FIGURE 11. Linear Function: $y = 3x + 100$	98
FIGURE 12. Residuals as a Percentage of the Observed Values	99
FIGURE 13. Percentage Difference between 1000 True Points and the Model:	99
FIGURE 14. Log Function: $y = 10\ln(x) + 10$	100
FIGURE 15. Residuals as a Percentage of the Observed Values	101
FIGURE 16. Percentage Difference between 1000 True Points and the Model:	101
FIGURE 17. 2nd Order Polynomial: $y = 2x^2 + 5x + 1000$	103
FIGURE 18. Residuals as a Percentage of the Observed Values	104
FIGURE 19. Percentage Difference between 1000 True Points and the Model:	104
FIGURE 20. Friedman's Fit of Pozzetti et al's "Signature"	107
FIGURE 21. New Fit of Pozzetti et al's "Signature"	108
FIGURE 22. Residuals as a Percentage of the Observed Values	109
FIGURE 23. Percentage Difference between 1000 True Points and the Model:	109
FIGURE 24. The Reactor	114
FIGURE 25. An ACE Service Handler	115
FIGURE 26. ASX Stream, Module, Task.	117
FIGURE 27. Message Broadcast Through ACE Event Server	119
FIGURE 28. Call Structure and Active Objects	120
FIGURE 29. Layered Queuing Model of ACE Event Server	123
FIGURE 30. MARS Resource Function for the Supplier Handler	126
FIGURE 31. Residuals as a Percentage of the Observed Values	127
FIGURE 32. MARS Resource Function for the Consumer Router	130
FIGURE 33. Residuals as a Percentage of the Observed Values	131

FIGURE 34. Linear Resource Function for the Consumer Router	133
FIGURE 35. Linear's Residuals as a Percentage of the Observed values	134
FIGURE 36. Comparing the Two Throughputs	136

List of Tables

TABLE 1. Correctness of EA: 30 Generated Functions, Intervals of 25 & 100 points	95
TABLE 2. Means from 30 Replications vs MARS Model Estimate, for 10 Cases	127
TABLE 3. Means from 30 Replications vs MARS Model Estimate, for 20 Cases	132
TABLE 4. Estimated Service Time, 200 Byte Message, 10 Consumers	135

1.0 Motivation

The purpose of modeling a real world system is not to achieve a perfect representation of that system but rather to build a model with “good enough” fidelity for the purposes of a given study. Fidelity, here, refers to how closely the details of the real world system have been represented in the model. A model of a computer system, for instance, which incorporates the computer’s virtual memory management system and cache would have greater fidelity than a model which did not. The accuracy of a model, on the other hand, refers to how closely it predicts the behaviour of the real world system. The required accuracy of a model will be a constraint imposed by the modeler and will depend on the type of study being performed. Obviously, the greater the fidelity of a

model, the more accurately it can predict the real system's behaviour. Incorporating too much detail in a model, however, may require a tremendous amount of effort and not render the model any more accurate than it needs to be. What we witness is a diminishing marginal return to the extra detail in the model. The goal of "good enough" fidelity, then, is to determine the minimum level of detail required to produce a model with the desired accuracy.

The problem of determining a "good enough" fidelity for a model is often encountered in the field of software performance modeling. Modern, distributed, software applications are complex systems which can, potentially, suffer from non-intuitive performance problems. Since the cost of developing these applications can be high and since a company's daily operations may rely on such systems, software performance modeling is being employed to discover performance problems throughout the development life cycle. However, since there are a myriad of potential details (from hardware and operating system features to compilation options and application-level parameters) a decision must be made as to what details should and should not be included in the model. This problem is particularly important in the early stages of development

since it is not desirable for the performance modeling to drastically slow down development.

A modeling technique which naturally lends itself to modeling client/server systems is the Layered Queuing Network model described in [Woodside95] and [Rolia95]. Layered queuing is described in more detail in Chapter 2, but in general it allows the software and hardware resources of a distributed application to be described as a set of tasks. Each task can offer a variety of services which are represented by task entries. An entry in one task can make requests to entries in other tasks. The fidelity of a layered queuing network model can be subdivided into its structural fidelity and its parametric fidelity. The number of tasks, the number of entries, and the calling relationships between the entries define the structure of a model. The structural fidelity of a model, therefore, refers to how closely the structure of the model resembles the structure of the actual application. For a given structure, the execution times assigned to each task entry make up the parameters of the model. The degree to which the model estimates of these parameters approach the actual values in the real system is called the parametric fidelity of the model.

One source for these execution time estimates is a software component's performance data sheet. This type of documentation, also referred to as a summary performance specification, was introduced by Booth and Wiecek ([Booth80]). It describes both the high-level functionality and the performance characteristics of a software component. With the advent of object-oriented frameworks and a growing awareness of the importance of software reuse, more applications are being built around existing components. Modeling applications composed of sub-components is greatly facilitated if the performance of those components has already been documented.

An essential part of the performance data sheet is the software component's resource function. Resource functions, discussed by Vetland and Woodside,[Vetland96], model the relationship between a software component's resource demand and the parameters which influence that demand. Most often, the resource demand refers to the CPU time consumed by a component (a fuller definition is provided in Chapter 2). To construct a resource function, the demand must be measured for various values of its independent parameters and a mathematical model is then built to fit the data.

Resource functions fit as submodels into the overall performance model by providing estimates for parameters like task entry execution time (see Chapter 2). The

execution time of a task entry will often depend on a number of component-specific parameters. In a sorting routine, for instance, the CPU demand will depend on the number of items to sort. Resource functions describe how the resource demand changes as the values of the independent parameters vary. They are particularly useful, therefore, when examining a performance model's behaviour as the variables which influence the resource demand of that model's sub-components change. If each task entry is a component, the entry execution times can be easily estimated using the component's resource function.

Since a resource function is itself a model, the same problem of determining a "good enough" fidelity is encountered when attempting to build a resource function. In this context, detail refers to the number of independent parameters considered, the various levels of parameters at which the measurements are made and the number of measurements. Assuming there is an underlying true function of the component's resource demand, the accuracy of a resource function is defined as the percentage difference between its predictions and the values of the "true" function. The question of "good enough" fidelity in a resource function reduces to a question of how many measurements of the resource demand should be made, at what levels of the independent

parameters should they be made at and how should the data be modeled so that the resulting function is within a specified accuracy.

The accuracy of a resource function directly impacts the parametric fidelity of the performance model in which it is used. As such, part of building a performance model with a “good enough” fidelity involves constructing resource functions which satisfy certain accuracy requirements. A technique, therefore is needed which can build a resource function to within a specified accuracy determined by the accuracy requirement of the performance model in which it is used. The fidelity of the resource function will be considered “good enough” if its accuracy falls within that limit.

1.1 An Illustration of a Resource Function

In their characterization of CPU demand for TCP/IP send and receive, Pozzetti et al[Pozzetti95] performed a full factorial experiment varying the message size in bytes from 64 to 5120 bytes in steps of 64 bytes, and the socket buffer size between 4k, 8k and 32k. The resulting “signatures” (i.e.resource demand characterizations) demonstrated discontinuous, non-linear behaviour and so no attempt was made to model them with

linear regression. One of their signatures, for TCP/IP send between two HP700's with a socket buffer size of 8k, is reproduced here to illustrate the type of data which may be encountered when creating a resource function.

FIGURE 1. Pozzetti et al's Signature

Pozzetti et al's experiment raises two issues: 1) How many levels of each parameter are needed - that is how many points should the measurements be taken at? and 2) How should the data be modeled? If a resource function is non-linear or discontinuous like the above signature, more points will be required to describe it than if it had been a simple, straight line. On the other hand, factorial experiments, like that performed by Pozzetti et al, are particularly susceptible to measurement overkill. In a 2-parameter factorial

experiment, if each parameter is tested at 50 levels, the total number of measurements required is 2500. Collecting so many points can be costly and is probably not necessary for providing accurate estimates of demand. Having collected the points, there is also the question of how to model them. If the points do not lie on one hyperplane, simple linear regression will not provide an adequate enough fit. In such a case, we will have no option other than inspecting the data visually and choosing a transformation to perform on the data which will permit a linear fit. However, if the resource function is in 4-dimensional space or higher (i.e. 3 or more parameters), visual inspection becomes impossible. Unless we know the “true” underlying resource function, how can we model it?

Prior to this thesis, a Performance Workbench [Vetland96] had been constructed to facilitate factorial experiments on software components. The resulting resource demand measurements could be stored and simple linear regression tools were provided to analyse the data. As has just been pointed out, however, factorial experiments and simple linear regression have their limitations. What is required is a more sophisticated technique for constructing resource functions.

1.2 Contributions of this Thesis

This thesis addresses the problem of how to construct a resource function with a “good enough” fidelity. In particular, the contributions of this thesis are:

- An extension to the Performance Workbench which includes a resource function database and a more powerful regression technique for handling non-linear and discontinuous resource functions.
- An automated, efficient (non-factorial) procedure which builds a resource function of “sufficient” accuracy by incrementally taking enough measurements of resource demand at various levels of its independent parameters so that the accuracy of the resulting mathematical function comes within a specified limit
- An outline for specifying that accuracy limit based on the requirements of the performance model in which the resource function will be used.
- An accuracy estimate, based on pseudo-confidence intervals, which is used to gauge the percentage deviation of the fitted resource function from the underlying “true” function.
- An application of the automated, incremental resource function construction technique against the components of the ACE Event Server.

1.3 Approach Taken in this Thesis

1.3.1 Type of detail considered when constructing a resource function

As previously mentioned, detail in the context of resource function construction refers to the number of independent parameters considered, the number of resource demand measurements made, and the various levels of the independent parameters at which the measurements are made. In this thesis it is assumed that the modeler knows the relevant independent parameters affecting a component's resource demand. This thesis, then, focuses on determining the necessary number and location of measurements to use when constructing a resource function.

1.3.2 Fitting the data

Even when enough measurements are collected, if the data is not fit correctly, the resulting mathematical function will not be very accurate. The limitation of simple linear regression has already been mentioned. The regression technique which was chosen for our purposes is called multivariate adaptive regression splines (MARS) and was developed by Friedman ([Friedman89] [Friedman91]). It is a simple but powerful tool for modeling linear and non-linear surfaces in multiple dimensions. One of its natural attributes is that the fit it provides improves as more data points are added.

1.3.3 Estimating the accuracy of a resource function

The estimated accuracy of a resource function is defined as the estimated percentage difference between the MARS model and the “true” function. Calculations used to estimate this accuracy are presented in Chapter 4. In general, a pseudo-confidence interval is used to predict the percentage deviation of the MARS model from the “true function”.

1.3.4 Setting the required accuracy of the resource function

If the performance model in which a resource function is going to be used is known at the time the resource function is constructed, then the accuracy requirements of the performance model can be used to set the required accuracy of the resource function. It is assumed that all of the error in the performance model’s predictions stems from parametric infidelity. Provided we know the elasticity of the performance model’s responses (e.g. throughput, utilization) with respect to its parameters, we can then determine how accurate the resource function needs to be. The elasticity of a performance model response with respect to a parameter describes, for a given percentage change in a parameter, what the corresponding percentage change in the response will be. If the

performance model responses are constrained to be within a certain percentage of the truth, then the elasticity of each response can be used to determine the required percentage accuracy of the resource functions.

If a component's data sheet is built without knowledge of the performance model, expert opinion will be required to set a reasonable accuracy requirement for the resource function. Nevertheless, if part of the documentation includes the resource function's estimated accuracy, when a performance model is constructed, the accuracy of the resource function in conjunction with the elasticity of the model responses can be used to determine what percentage error the resource function contributes to the performance model.

1.4 An Incremental Procedure for Constructing Resource Functions

The result of this research, then, has been the development of a MARS-based incremental process which automatically determines the required level of detail in the construction of a resource function. MARS was incorporated into a higher-level, Tcl script called "smartexec", which incrementally chooses points, performs the measurements, models the data with MARS, and if the accuracy of the model is not

sufficient, takes more measurements. The procedure progresses towards the required accuracy and, in this way, is able to construct resource functions with “good enough” fidelity. The script was integrated into the Performance Workbench which now allows the user to run either a standard factorial experiment or an incremental one.

1.5 Overview of Chapters

Chapter 2 begins with an explanation of Layered Queuing and demonstrates how resource functions fit into the overall performance model. It also lays out a notational framework to describe how the fidelity of a resource function filters up to affect the accuracy of the performance model. It then provides an overview of previous work done related to resource functions. Finally it highlights some principles distilled from the field of experimental design which can be applied to resource function construction. Chapter 3 is entitled “MARS”. Beginning with a review of general linear regression and splines, it is devoted to describing how the MARS procedure works. Chapter 4 spells out the details of the incremental resource function construction process. Chapter 5 describes the current implementation of the “smartexec” script and presents the results of several validation tests. The idea was to see how well “smartexec” performed in modeling a variety of

known, underlying functions - linear, log, second order polynomial and discontinuous. Finally, Chapter 6 reports the results of a case study in which “smartexec” was used to construct the resource functions for the components of the ACE Event Server.

2.0 Overview

This chapter examines how the role of resource functions in performance modeling can be used as a means of guiding the resource function construction process. Understanding the context in which a resource function will be employed allows us to determine an appropriate level of accuracy for it. This chapter begins by providing some definitions to clarify what is meant by resources, resource demands, software components and resource functions. An overview of Layered Queuing is then presented along with a discussion on how the accuracy of a resource function affects the performance model's predictions. Following that, a brief introduction to the field of experimental design is offered to highlight the principles behind exploring an unknown surface. Finally, some

recent research related to resource demand characterization is examined with a view to understanding how the data was collected and modeled. This chapter, therefore, attempts to lay the groundwork for the main contribution of this thesis - an automated procedure for constructing resource functions with “good enough” fidelity.

2.1 Definitions

- **Resource:**

A resource is any entity in the system, whether software or hardware, which offers a service. The most common resource referred to, and the one which we will focus on during this thesis, is the CPU. Other resources are disks, memory, file servers, or even application-level servers.

- **Resource Demand:**

Resource demand quantifies how much a resource is used. It could refer to the number of requests made to a resource or to the amount of execution time the resource must spend servicing the request. Resource demand does not refer to the amount of time spent waiting to use a resource. This thesis concentrates on CPU time as the resource demand of interest. Other resource demands could be the number of calls to a server or the number of I/O operations performed on a disk.

- **Software Component:**

A software component can be any software entity(function, method, process, system) which performs a logical function. So a software component is a resource too. Since this thesis employs a performance model to guide resource function construction, a software component will most often refer to a task entry (see “2.2 Layered Queuing” on page 18). In order to perform its function, a software component must use other resources (like the CPU) and therefore it imposes a demand on those resources.

- **Resource Function:**

Typically, the resource demand of a software component will depend on a number of parameters. As an example, the CPU time required to send a message on a socket will obviously depend on the size of the message. The purpose of a resource function, therefore, is to capture how the resource demand of a software component changes with varying values of the independent parameters. A mathematical function is used as a compact representation for a resource function. What is important, however, is not so much how the resource function is represented, but how accurately it estimates the resource demand for a given set of parameter values.

- **Results, Responses and Parameters:**

Throughout this thesis, the term *result* will refer to the dependent variable of a resource function, i.e. the resource demand. The term *response* may be used either to signify a *result* or to describe an output, like throughput, from a performance model. The term *parameter* will refer to either to an independent variable in a resource function or to an input variable, like execution time, in a performance model. The context in which a term is used should resolve any ambiguity.

2.2 Layered Queuing

Large, distributed software applications may involve several layers of servers. A client may send a request to a server which, in turn, may require the services of a lower level server in order satisfy that request. The middle level server blocks until the lower level server completes; and the client blocks, waiting for a reply from the middle level server. This type of interaction, where a process makes a request for service and blocks until it receives a reply, is referred to as a rendezvous. Rendezvous' are prevalent throughout client/server systems but conventional queuing models, which were designed to model hardware contention, cannot properly estimate waiting times due to software

server contention. The Layered Queuing Network ([Woodside95],[Rolia95]), was developed as a means of properly modeling client/server systems with rendezvous interactions.

In a layered queuing model, both the software and hardware resources of a system are modeled as tasks. Tasks can send requests to each other. A task which only sends requests but never receives any is called a pure client; a task which both sends and receives requests is referred to as an “active server”; while a task which only receives requests is called a “pure” server. Tasks can offer more than one service. The performance characteristics of a service which it offers may also vary depending on the client making the request. Task entries are used to distinguish between the various services offered and the various performance characteristics of these services. The entries retrieve requests from a common FIFO queue. A task with entries, therefore, can be reduced to a single server with multiple classes of customers. A task can also be turned into a multi-server by setting its multiplicity to greater than 1. Graphically, a task is depicted as a parallelogram and, depending on the notation, an entry is depicted as a parallelogram placed inside or adjacent to the task. Arcs between tasks are used to describe calls.

The execution of each task entry is divided into phases. When a task sends a request to the entry of another task, the calling task is blocked until the receiving task retrieves the message off the queue and the appropriate entry completes its first phase of execution. Any subsequent phases of execution continue in parallel with the calling task. A call can be made completely asynchronous by having no execution time in the first phase of the entry. Using first phase allows the model to represent rendezvous interactions in client/server system.

Each task in a model is assigned to a processor. A processor task is automatically generated for each processor used. The execution time for each entry of each task in each phase is divided into slices, or quanta. An entry in the processor task is created to represent the quantum of execution time assigned to each phase of execution of each entry running on that processor. The execution time of an entry in a particular phase, therefore, is translated into a number of calls to the corresponding quantum entry in the local processor task. In this way, hardware contention can be modeled.

2.3 Impact of Resource Function Accuracy on the Performance Model

Assuming the modeler has correctly identified the various tasks and calling structure, the accuracy of the performance model will depend on the execution time estimates provided for the phases of each entry. Resource functions are used to provide these estimates. Figure 2 on page 22 depicts how the execution time estimates flow from the resource function into the performance model. The accuracy of the layered queuing model, therefore, directly depends on the accuracy of the resource function. In this section we provide a means of quantifying this dependency.

FIGURE 2. Resource Function Submodel within Performance Model

2.3.1 Notation

- Let r represent a response from the performance model such as a class throughput or the utilization of a task.
- Let Y be a vector of the performance model parameters, $Y=[y1,y2,y3...]$
- The model defines the relationship between Y and r . Let M be the function such that $r = M(Y)$
- Let yp be a model parameter (within the Y vector). Suppose yp represents the execution time of a task entry. The value of yp will itself be dependent on a number of parameters specific to the component whose execution time yp is estimating. Let X be the vector of parameters on which yp depends, $X=[x1,x2,x3...]$.
- A resource function defines the relationship between X and yp . Let f_{est} be the function such that $yp=f_{est}(X)$.

- The sensitivity of r with respect to one of the model parameters, yp , is

$$sens(r, yp) = \frac{\partial M(Y)}{\partial yp}$$

- The elasticity of r with respect to one of the model parameters, yp , is

$$elas(r, yp) = \frac{\partial M(Y)}{\partial yp} \cdot \frac{yp}{M(Y)}$$

The elasticity of r describes, for a given percentage change in yp , the corresponding percentage change in r .

2.3.2 Quantifying the impact of resource function error on performance model accuracy

The resource function, $f_{est}(X)$, is only an approximation to the “true” function, $f(X)$, relating the component’s resource demand to its independent parameters. Suppose, at any point along its curve, $f_{est}(X)$ differs by no more than $PercentageError(f_{est}(X))$ percent from $f(X)$. Since $f_{est}(X)$ provides an estimate for the performance model parameter yp , the percentage error in the performance model response, r , induced by an estimate error in yp is:

$$PercentageError(r) \cong PercentageError(f_{est}(X)) \cdot elas(r, yp)$$

Using this fact, we can also set the required accuracy on a performance model response and work backwards to determine the necessary accuracy of the resource function.

2.3.3 Determining the Elasticity of a Performance Model Response

It has been proven, [Gordon80], that the elasticity of a response in a closed multiclass queuing model is not necessarily less than 1 with respect to an execution time. If the elasticity was less than 1, we could simply state that a tolerable percentage error in a performance model response requires that the resource functions be accurate to within the same percent. There do exist elasticity equations which can be used for certain classes of

queuing models, but not for layered queuing models . Sheikh [Sheikh97], however, has been able to extract model sensitivities as the model is solved. These can then be used to calculate point elasticities for the given set of performance model parameter estimates. A cruder alternative would be to manually perturb the execution time values and observe the effect on the performance model response.

2.4 Constructing Resource Functions

To construct a resource function , an experiment must be designed and measurements must be taken. The number and location of measurements in addition to how the measurements are modeled will determine the accuracy of the resource function. In this section we examine some principles of experimental design and some recent research in resource function construction.

2.4.1 Principles of Experimental Design

When measuring resource consumption values, the choice of parameter values for the measurements is called the experimental design, and each combination of parameter settings is called a design point. The resource consumption values form a response surface. There is, in fact, a whole field of research devoted to experimental

design and response surface methodologies[Myers89]. A response surface captures how the value of a dependent variable (the response or the result) varies as the values of some independent variables (the parameters) change. The main idea behind these methodologies is that an experiment is designed in order to explore and to model some limited area of the response surface which the modeler is interested in. The design itself should be based on what form the modeler assumes the surface to have. For example, a simple response surface which can be modeled as a first order linear hyperplane requires fewer points to be collected than a more complicated second order model.

The underlying assumption is that the modeler knows, approximately, what the response surface looks like in the area of interest. And if the modeler does not know, then it is assumed that he/she will engage in an iterative process[Box87]: collecting points, modeling them, assessing the goodness of fit, and if the fit is unsatisfactory, changing the functional form of the model and collecting any additional points to satisfy that form. The question still arises though as to what the functional form should be. In multiple dimensions, it will not always be obvious what model should be tried next if the first order linear model does not provide a good fit. Nevertheless, the notion of an iterative process is useful and it provides the basis for the method described later in the thesis.

As a minimum, when the modeler is uncertain of the form of the response surface, [Box87] suggests beginning with a first order fit and a 2^K factorial design, where K is the number of independent parameters. In such a design, two levels of each parameter are considered and a design point exists for each combination of parameter values. If three parameters are involved and each parameter can take on one of two values (e.g. -1 and +1), the design can be visualized as the following cube in Figure 3 on page 27:

FIGURE 3. Visualizing a 2^K factorial design with three parameters

Box's proposal is to start with a simple 2^K factorial experiment, model the data using least squares (see Chapter 3), with a first order polynomial, and if the fit is not adequate, move to a more complicated model, such as a second order polynomial.

There are a number of suggested experimental designs to use when striving for a second order fit [Khuri87]. One of them is the 3^K factorial design which is the same as the 2^K except that three levels for each parameter are considered. The even dispersal of points throughout the design space is used to minimize the variance of the estimated response values in the least squares model. The second order polynomial is not the only alternative. A log transformation can be used on data which follows a geometric form. In such a case, the experiment is designed so that parameter values increase by powers of 2. In general, the experimental design is chosen to suit the assumed form of the “true” response surface.

From this discussion, we can extract two principles to help us in constructing resource functions:

- The process must be iterative, beginning with a simple model and progressing, if necessary towards a more complex model
- More complex models will require more points

The iterative process just described, however relies on the intuition of the modeler to choose the correct functional form to use. If the modeler is inexperienced or the data does not lend itself to a standard functional form, then the process will not converge towards an

accurate fit. The problem is exacerbated in multiple dimensions when visual inspection of the data is impossible. What is needed is a procedure which automatically chooses the functional form to use and whose fit improves as more points are added to the design. As it turns out, there does exist a non-parametric, spline-based regression technique, MARS, which satisfies that need (Chapter 3).

2.4.2 Recent Work in Software Resource Demand Characterization

Although the term “resource function” is not in general use, the concept of characterizing the resource demand of a software component at various parameter levels can be found in a number of articles. One of the earliest references is [Booth80], where a “cost metric” is used to describe the number of CPU cycles consumed by an operation in an abstract data type as the input to that operation changes. We have already mentioned Pozzetti et al’s TCP/IP signatures[Pozzetti95]. In a similar study, Xu and Hwang[Xu96] modeled the communication overhead of message passing on an IBM SP2, under different combinations of message size and number of nodes involved in group communication. Vetland [Vetland93] referred to resource functions as “complexity functions” and used them to capture CPU, disk, and network demand for the components in a variety of distributed applications. Vetland and Woodside[Vetland96] captured the resource function

for XDR marshalling overhead in remote procedure calls, varying the length of opaque data. A practical use for these resource functions is found in Menasce's CLISSPE performance modeling script language[Menasce97] which expects, as input, linear models relating the CPU demand to the number of data bytes involved in RPC and I/O calls.

Booth and Wiecek attempted to expand the traditional abstract data type to include performance information. The resulting "performance abstract data type" contained "cost metrics" for each operation of the ADT which described how the CPU demand of an operation was related to the inputs for that operation. Constructing these functions required an in-depth analysis of the control flow and data flow within an operation. Each operation was divided into primitive sub-operations and the average number of times a primitive sub-operation was called was determined by examining the size of the data structures involved and the probability of transition from one primitive sub-operation to the next. The authors suggested that a library of performance ADTs could then be developed and used to evaluate the performance characteristics of alternative system designs - where each system is built on top of the existing ADTs. Although the notion of a library of resource functions is useful, the proposed construction

technique for the “cost metrics” requires a prohibitive amount of work, not to mention the availability of source code.

[Vetland93] suggests a more practical approach to building resource functions which involves measuring the component’s resource demand at various levels of its independent parameters and fitting the measurements with regression. Source code is used only as a means of determining the mathematical form of the regression fit (e.g. linear, polynomial, log etc.). Vetland writes:

“...the form of each complexity function should be determined by code walk-through or possibly by inspection of the design documents for the software components. Thus, white-box measurement experiments are considered necessary” (p 204)

When source code is not available, the modeler must resort to black-box testing, in which case Vetland suggests a manual process of “iterated assumption and testing against the data” (p. 203) - i.e. taking measurements, guessing at the proper form of the function, modeling the data with that form, and if the fit is not good enough, changing the form and possibly collecting more data. Vetland himself concedes that this process is “very difficult and time-consuming to undertake”. It is often the case, however, (e.g. when characterizing operating system components) that the source is not available. What is needed then is an automated process which can iteratively progress towards an accurate

enough form for the resource function. The notion of an accurate enough resource function is not dealt with at all in [Vetland93] and no attempt is made to show the impact of complexity function inaccuracies on the performance models in which they are used.

Pozzetti et al demonstrate some concern with the accuracy of their “signatures” by constructing confidence intervals around each measurement. Since there was no overriding performance model fidelity constraints, though, no justification was given for the tightness of the confidence intervals they achieved. Furthermore, since the signatures were discontinuous and non-linear, no attempt was made to model them with a mathematical function.

An issue raised in both [Pozzetti95] and [Vetland96] is that of experimental design. Pozzetti et al performed a factorial experiment varying message size from 64K to 5120K in steps of 64K and varying socket buffer size between 4K, 8K, and 32K. Vetland and Woodside varied the length of opaque data in remote procedure calls between 20 and 500 bytes in steps of 20 bytes. In both cases, their purpose was simply to explore the resource demand characteristics and so little explanation was provided for the choice of their experimental design. A practical resource function, however, is constructed for a purpose, and that purpose should guide the experimental design. That is, the required

accuracy of the resource function and the complexity of the underlying “true” function should determine how many measurements are needed.

In [Xu96], the situation is somewhat different because the authors assumed, apriori, what the form of the resource function would be. The ranges of their parameters were determined by the limits of the application and they employed a standard practice of increasing the parameter values by powers of 2 to cover that range. This practice, however, can only be justified if you know, more or less, the form of the underlying function. By giving more attention to the lower parameter values, some of the signatures provided by Pozzetti, for instance, would not have been adequately captured using this approach.

Current research, therefore, has not provided a practical means for constructing resource functions, especially when source code is unavailable. Little attention has been given to what type of experimental design should be used during black-box testing when the shape of the underlying, “true”, function is unknown. In addition, there remains the nagging problem of how to model the data once it is collected. If it does not fit the standard first order, what model should be used? Menasce’s modeling language, for instance, will only accept linear resource functions, regardless of how well those

models fit the actual data. Furthermore, there has been no discussion on the impact of resource function inaccuracy on performance model predictions nor has there been any attempt to construct resource functions to within a required accuracy.

2.5 Summary

The final deliverable of the resource function construction process is a mathematical model which compactly describes the behaviour of the result variable with respect to the parameters. Since the resource function is used within a performance model, the accuracy requirements of the performance model should be used to set the accuracy requirements of the resource function. The elasticity of the performance model response variables with respect to the model parameters can be used to set this accuracy constraint. Current research has not adequately dealt with the questions of how to set the required accuracy of the resource function nor how to construct it to within that accuracy. In this thesis, we draw on the notion, from the field of experimental design, of an iterative construction process. Instead of relying on standard functional forms, a non-parametric spline-based regression technique (MARS) is used, whose fit improves as more points are

2.5 Summary

added to the design. The procedure described in this thesis progresses towards the required accuracy set by the performance model. In this way, we can automatically construct resource function with “good enough” fidelity.

MARS (Multivariate Adaptive Regression Splines)

3.0 Overview

MARS ([Friedman91]) was chosen to model the resource demand measurements collected during an experiment. It is a non-parametric technique that does not use standard functional forms to fit the data, relying instead on regression splines. In this way it can fit linear, non-linear and even discontinuous response surfaces. The purpose of this chapter is to present the necessary mathematical background for understanding MARS and then to describe the actual MARS procedure. For convenience, this chapter begins with some notation which can be referred to throughout the thesis. Section 2 describes general linear regression. Section 3 discusses splines. Section 4 examines regression splines. Section 5 presents the MARS procedure.

3.1 Notation

- Let y be the result (dependent) variable
- Let K be the number of independent parameters
- Let X be a vector of independent parameters $X=[x_1, x_2, \dots, x_K]$
- Let $f()$ be the “true” resource function relating y and X , $y=f(X)$
- Let y_o be the observed values of y in an experiment
- Let e be a random measurement error
- Let N be the number of observations
- Let X_i , $i=1..N$, be a design point in an experiment
- Let y_{o_i} , $i=1..N$, be the observation at the i th design point
- Let x_{p_i} , $p=1..K$, $i=1..N$, be the value of the p th independent parameter at the i th design point
- Let n_p , $p=1..K$, be the number of distinct values of the p th independent parameter
- Let $f_{est}()$ be an estimate of $f()$, determined by regression
- Let $f_{est}^{-i}()$, $i=1..N$, be an estimate of $f()$, determined by regression, not including the i th design point
- Let w_i be the weight assigned to the i th design point during regression
- Let B , be the number of basis functions in a regression model
- Let $t_j(X)$, $j=1..B$, be one of the basis functions
- Let $tv(X_i)$, $i=1..N$, be the $1 \times B$ vector of the values of each basis function at X_i ,
 $tv(X_i)=[t_1(X_i), t_2(X_i), \dots, t_B(X_i)]$
- Let TM be the $N \times B$ matrix where the i th row is $tv(X_i)$
- Let a_j , $j=1..B$, be the coefficient which regression assigns to the j th basis function
- Let df be the degrees of freedom used up by a regression model
- Let s be the smoother used in the MARS fit

3.2 On General Linear Regression

An excellent description of applied linear models can be found in[Neter74].

Suppose, in our experiment, there is a result variable y which is dependent on K parameters, $x_1, x_2, \dots, x_K$. We assume there is a “true” function, f , describing the relationship between the result variable and the parameters:

$$y = f(X) \quad X = [x_1, x_2, \dots, x_K]$$

Due to random measurement errors, the actual observations, y_o of y , can be expressed as follows:

$$y_o = f(X) + e$$

where e represents a random error. The goal of regression is to provide a reasonably good estimate for $f(X)$, call it $f_{est}(X)$. General linear regression represents $f_{est}(X)$ by a linear combination of known functions. That is:

$$f_{est}(X) = \sum_{j=1}^B a_j t_j(X)$$

where each a_j is a coefficient, each $t_j(X)$ is a function, or transformation, of the parameters, and B represents the total number of such transformations. The $t_j(X)$ functions are

referred to as basis functions. In the simplest case, $f(X)$ might be approximated with a constant basis function in which case $B=1$, $t_1(X)=1$, and $f_{est}(X) = a_1$.

Given a set of basis functions and a set of points, the purpose of general linear regression is to calculate $f_{est}(X)$ by determining the coefficients a_j , which minimize the sum of squared differences between the real response values and the those estimated by $f_{est}(X)$. That is, regression attempts to minimize:

$$\sum_{i=1}^N (f(X_i) - f_{est}(X_i))^2$$

If some of the points have more importance or likelihood of occurring (or have errors of different importance), a set of weights, w , can be provided for the points. In this case, a weighted least squares is performed which attempts to minimize:

$$\sum_{i=1}^N w_i (f(X_i) - f_{est}(X_i))^2$$

In either case, however, the modeler must provide the individual transformation functions $t_j(X)$, $j=1..B$.

3.2.1 Errors, Cross Validation, and Generalized Cross Validation

When using regression to build a resource function it is important to have some notion of the model's error in order to gauge the accuracy of the resource function. A common measure of the model's error is the mean squared error(MSE), which is simply the sum of squared residuals divided by the remaining degrees of freedom after the model has been fit to the data i.e.:

$$MSE = \frac{\sum_{i=1}^N (y_{o_i} - f_{est}(X_i))^2}{N - df}$$

where N is the number of data points, df is the degrees of freedom used up by the fit, X_i is a design point, y_{o_i} is the observation at that design point and $f_{est}(X_i)$ is the model's estimated response at that design point. MSE is an estimate of the variance of the residuals.

The future squared error of the model is related to MSE, but instead of measuring the model's deviation from the observed points, it estimates what the model's deviation would be from a new set of measurements. That is, it provides a metric for the

model's predictive capability. One means of estimating the future squared error is through cross validation. The cross validation value for a model is calculated as follows:

$$CV = \frac{1}{N} \cdot \sum_{i=1}^N (y_{o_i} - f_{est}^{-i}(X_i))^2$$

where $f_{est}^{-i}()$ is the regression model recalculated with the i th point missing (this notation follows that provided in [Friedman89]). In other words, it calculates how well a regression model derived without the i th point would predict the i th point. Since each term in the sum requires a complete recalculation of the of the regression model, the CV score is usually too expensive to compute. [Wahba90], however, introduced a cheaper approximate called generalized cross validation, GCV. GCV is calculated as:

$$GCV = \frac{1}{N} \cdot \frac{\sum_{i=1}^N (y_{o_i} - f_{est}(X_i))^2}{\left(1 - \frac{df}{N}\right)^2}$$

The term:

$$\frac{\sum_{i=1}^N (y_{o_i} - f_{est}(X_i))^2}{N}$$

is referred to as the average squared residual, or ASR. So GCV is the average squared residual inflated by a denominator dependent on the number of points and the degrees of freedom used up in the fit. When rearranged, GCV is just an inflated value of MSE. i.e.

$$GCV = MSE \cdot \frac{N}{N - df}$$

GCV, then, is an estimate of the variance of residuals from future measurements against the current regression model. Wahba has shown that GCV approaches CV asymptotically as N is increased to infinity.

3.2.2 Confidence Bands

Ideally we would like to know the model's deviation from the "true" function, $f(X_i)$. Although $f(X_i)$ is unknown, confidence intervals around the regression model can be used to estimate the region in which the true function lies for a given level of confidence. Provided that residuals are normally distributed and homoscedastic (constant variance), the variance of the model estimate at any point X_i is calculated as:

$$var(f_{est}(X_i)) = MSE(tv(X_i) \cdot (TM TM)^{-1} \cdot tv'(X_i))$$

If $f_{est}()$ is defined as a linear combination of B basis functions, $t_1()$, $t_2()$, ... $t_B()$, then $tv(X_i)$ is a vector defined as:

$$tv(X_i) = [t_1(X_i) \ t_2(X_i) \ \dots \ t_B(X_i)]$$

And TM is the matrix of these vectors for all N points. i.e.

$$TM = \begin{bmatrix} tv(X_1) \\ tv(X_2) \\ \dots \\ tv(X_N) \end{bmatrix}$$

And TM' and $tv'(X_i)$ are the transposes. The confidence interval at any point, then, can be calculated as follows:

$$f_{est}(X_i) \pm \sqrt{var(f_{est}(X_i))} \cdot z(1 - \alpha/2)$$

where z is the z -statistic for the normal distribution and α is the probability that the true value does not lie in this region.

It is not always the case, however, that the assumptions of normality and homoscedasticity hold. A more general approach to calculating confidence intervals is the bootstrap technique. Bootstrapping, see [Efron79], [Efron81] involves re-sampling

errors from the observed residuals, re-generating response values, and re-constructing the model. Each time the model is reconstructed, a different response will be estimated at a given design point. By doing this a number of times, a distribution of estimates will be observed at each design point. The histogram of estimates at each point is then used to define the confidence interval. In general the procedure is as follows:

- Given N measurements, fit the data with a model, $f_{est}(X)$
- For each residual e_i , $i=1\dots N$, assign a probability of $1/N$
- Randomly sample (with replacement) N residuals from the observed errors, call them $e_i^{resample}$, $i = 1\dots N$
- Regenerate a set of responses,

$$Y_i^{resample} = f_{est}(X_i) + e_i^{resample} \quad i = 1\dots N$$

- Reconstruct the MARS model, call it $f_{est_1}^{resample}(X)$
- Repeat the above 3 steps r number of times so that a series of MARS models are constructed: $f_{est_1}^{resample}(X) \dots f_{est_r}^{resample}(X)$
- At each design point, r number of estimates will be provided
- To obtain a 95% confidence interval at a design point, order the estimates from least to greatest. By dropping the first and last 2.5% percent of the estimates, the remaining values span the entire width of the interval.

The obvious disadvantage to the bootstrap technique is the computation involved.

[Efron81], for instance, suggested using 1000 replications when calculating confidence intervals.

Neither technique for deriving confidence intervals, therefore, is perfect. The standard, parametric approach relies on assumptions which may not hold. Bootstrapping, on the other hand, simply requires too much work. The compromise adopted in this thesis, for the sake of computational efficiency, is to use the parametric approach but replace MSE with GCV, use 3 standard deviations instead of $\sqrt{(1 - \alpha/2)}$ and use the widest width along the confidence band as the width for the entire confidence band. Furthermore, the confidence interval calculation is performed in the log domain and then transformed back to the real domain in order that the confidence band may be based on percentage deviation instead of absolute residual. The resulting “pseudo-confidence” interval will be wider than the MSE-based one without requiring a prohibitive amount of computation to calculate. Chapter 4 describes the calculations in greater detail.

3.2.3 A Problem with Regression Modeling

The obvious drawback to general linear regression is that the modeler must know, in advance, what the basis functions should be. If the underlying function is not known, visual inspection of the points can suggest a possible curve, but in high dimensions this becomes difficult. The benefit of MARS is that it determines, approximately, what basis functions to use. The basis functions which it constructs are

products of truncated power functions and the model returned by MARS is a truncated power spline. In order to understand how MARS works we should understand a bit about splines.

3.3 On Splines

It is a well known fact (e.g. [Prenter75]) that any set of $(N+1)$ points in two dimensions can be interpolated by a Lagrangian polynomial of degree N . With such a function, however, as N increases, its behavior in between the points can vary drastically from that of the true function. In practice, instead of fitting the points globally with one polynomial, a number of simpler, smaller-ordered functions can be used to fit points in a certain interval. The points at the boundaries of each interval, where the functions are joined, are referred to as knots and the resulting approximation is known as a spline. Splines are used to approximate functions whose true natures are unknown. In the simplest case, if we confine the spline to consist of first order polynomials, each pair of points can be interpolated by 1 polynomial. In general, a spline is a function which combines a set of piecewise functions obeying certain continuity constraints at the knots.

Mathematically, one way to describe a spline which is continuous at each knot is through the use of cardinal basis functions(e.g. [Prenter75]). A cardinal basis function takes the following form:

$$c_i(x) = \begin{cases} 0 & x \leq x_{i-1} \\ \frac{(x - x_{i-1})}{(x_i - x_{i-1})} & x_{i-1} \leq x \leq x_i \\ \frac{(x_{i+1} - x)}{(x_{i+1} - x_i)} & x_i \leq x \leq x_{i+1} \\ 0 & x \geq x_{i+1} \end{cases}$$

Graphically, it reaches a maximum of 1 at x_i and falls to 0 at the surrounding points. i.e.:

The equation for such a spline then, is a linear combination of these cardinal basis functions:

$$f_{est}(x) = \sum_{i=1}^N y_{O_i} c_i(x)$$

where N is the number of points, and each coefficient, yo_i , is just the value of the dependent variable at point i . Such a spline interpolates linearly between the points and is continuous at each knot. In this case, each point is a knot.

3.3.1 Truncated Power Splines

A more compact basis function is the truncated power basis function[Friedman89]. A truncated power function is defined as:

$$tpf_i(x) = (x - x_i)_+^{pow} = \begin{cases} 0 & x - x_i \leq 0 \\ (x - x_i)^{pow} & x - x_i > 0 \end{cases}$$

where x_i is a knot, pow is the power of the function. MARS constructs its model out of these types of basis functions and since it only considers truncated power functions with powers of one, for the rest of this discussion we will assume $pow=1$. Graphically such a functions, with $pow=1$, looks like:

For a given set of N points in two dimensions, the truncated power spline, with $pow=1$, is defined as:

$$f_{est}(x) = a_0 + \sum_{i=1}^{N-1} a_i(x-x_i)_+$$

where a_1 is the slope of the first segment and the subsequent a 's represent the change in slope from the previous segments. As an example, consider the set of points $(x,y_0): \{(1,1) (2,3) (3,8)\}$. A truncated power spline which interpolates these points could be:

$$f_{est}(x) = 1 + 2(x-1)_+ + 3(x-2)_+$$

Regardless of the coefficients it has, it is guaranteed to be continuous at the knots. To see this, consider the limit of the following spline as the knot $x_l, x_l \geq x_k$ is approached from either side:

$$\lim_{x \rightarrow x_l} a_0 + a_1(x-x_k)_+ + a_2(x-x_l)_+ + \dots$$

In either case, the limit will obviously be: $a_0 + a_1(x_l-x_k)$.

The truncated power functions we have seen so far only return positive values for points larger than the knot. An equally legitimate truncated power function can be defined so that it only returns a positive value for points less than the knot. That is:

$$tpf_i(x) = (-(x - x_i))_+^{pow} = \begin{cases} 0 & x - x_i \geq 0 \\ (-(x - x_i))^{pow} & x - x_i < 0 \end{cases}$$

We will call this the complement form of a truncated power basis function. Graphically the complement looks like:

One technique which MARS employs is to create 2 truncated power functions around a single knot. If it determines that x_i should be a knot, the 2 functions it adds to the model are: $(x - x_i)_+$ and its complement $(-(x - x_i))_+$. Using this method, the three points discussed above could be interpolated with the following spline:

$$f_{est}(x) = 3 - 2(-(x - 2))_+ + 5(x - 2)_+$$

3.3.2 Truncated Power Splines in Multiple Dimensions

Truncated power splines can also be used to interpolate multidimensional data.

Consider the 3-dimensional case where the result variable y is dependent on the parameters x_1 and x_2 . Suppose there are n_1 values of x_1 and n_2 values of x_2 . If we fix x_2 at a given value, the truncated power spline interpolate for the response values is:

$$a_0(x_2) + \sum_{i=1}^{n_1} a_i(x_2)(x_1 - x_{1_i})_+$$

$a_i(x_2)$ is a function whose values depend on the level of x_2 . The different values of $a_i(x_2)$ can themselves be interpolated with a truncated power spline. That is:

$$a_i(x_2) = a_{i0} + \sum_{j=1}^{n_2} a_{ij}(x_2 - x_{2_j})_+$$

So the overall truncated power spline is:

$$a_{00} + \sum_{j=1}^{n_2} a_{0j}(x_2 - x_{2_j})_+ + \sum_{i=1}^{n_1} \left(a_{i0} + \sum_{j=1}^{n_2} a_{ij}(x_2 - x_{2_j})_+ \right) (x_1 - x_{1_i})_+$$

or:

$$a_{00} + \sum_{j=1}^{n_2} a_{0j}(x_2 - x_{2_j})_+ + \sum_{i=1}^{n_1} a_{i0}(x_1 - x_{1_i})_+ + \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} a_{ij}(x_2 - x_{2_j})_+(x_1 - x_{1_i})_+$$

This particular representation does not consider truncated power functions in the complement form.

In high dimensions, with many points, a truncated power spline which interpolates the data will contain a huge number of terms. The main idea behind MARS is to select a subset of these terms and to use those as basis functions. The actual constants, a_{ij} , are determined by performing a regression over the selected basis functions. Before describing MARS fully though, a note on regression splines is appropriate.

3.4 On Regression Splines

Up to this point, we have only considered models which interpolate the data. Often, however, the random error introduced into measurements makes interpolation undesirable. What is desirable is a modeling technique which approximates the true function without being constrained to fit each measurement. Regression splines attempt to do just that. When building regression splines, certain points are chosen as knots. Basis functions are then constructed around these knots and regression is performed over the basis functions to determine the best coefficients. Notice that the goal is not even to interpolate the responses at the chosen knots. The only requirement is that the spline be

continuous over the entire region. Consequently, a regression spline is less vulnerable to random measurement error and, therefore, smoother than an interpolating spline. The classic approach to regression splines is to place knots at fixed locations throughout the parameter space. MARS improves on this process by adaptively choosing the knots.

3.5 Multivariate Adaptive Regression Splines - MARS

The MARS procedure introduced in [Friedman91] is based on recursive partitioning. That is, it adaptively partitions the data into regions and performs regression over the individual regions. Regions can be further subdivided if the regression model for that region does not provide an adequate fit. Truncated power functions are defined based on the chosen boundary points. If there is only 1 parameter, x , and MARS partitions the data at knot x_k , the two regions can be represented by two basis functions: $(x-x_k)_+$ and $(-(x-x_k))_+$. The first provides nonzero values for all points greater than x_k and the second provides nonzero values for all points less than x_k . When regression is performed over these basis functions, each region will be modeled with a different line and the lines will join at x_k . With more than one parameter, MARS can choose to partition along any of the parameters.

Truncated power functions are also used to represent further subdivisions of a region. If the data is initially partitioned along parameter xi at knot xi_k , then the spline model will contain the terms: $(xi-xi_k)_+$ and $-(xi-xi_k)_+$. Now suppose MARS determines that all points with an xi value greater than xi_k should be further partitioned along parameter xj at a knot value of xj_k . To represent this new subdivision, two new basis functions will be added to the model: $(xi-xi_k)_+(-(xj-xj_k))_+$ and $(xi-xi_k)_+(xj-xj_k)_+$. The first only provides an effect for points with an xi value greater than xi_k and an xj value less than xj_k while the second provides an effect for points in the same xi range but whose xj values are greater than xj_k . These two basis functions introduce an interaction effect into the model. Within the newly divided region, the effect of xi on the result value is dependent on the level of the xj parameter.

In general, MARS is an iterative procedure which, on each iteration, scans through the existing regions (represented by the basis functions built from the previous iteration) and finds the next best knot at which to partition the data. When a region is partitioned, the old basis function representing that region is not discarded from the model. The procedure begins with a simple model consisting of the constant basis function, 1, which represents the entire set of points. On the first iteration, this region will

be split, resulting in a model with three basis functions: 1, $(x_i - x_{ik})_+$ and $-(x_i - x_{ik})_+$. On subsequent iterations, the whole region, or any of the new subregions, may be selected for partitioning. After the second iteration, for instance, the basis functions may look like: 1, $(x_i - x_{ik})_+$, $-(x_i - x_{ik})_+$, $(x_j - x_{jk})_+$ and $-(x_j - x_{jk})_+$ or: 1, $(x_i - x_{ik})_+$, $-(x_i - x_{ik})_+$, $(x_i - x_{ik})_+(-(x_j - x_{jk})_+)$ and $(x_i - x_{ik})_+(x_j - x_{jk})_+$. The partition boundaries are implicit in the knot values. In effect, what MARS does is to incrementally build a model by selecting, as basis functions, a subset of the terms from the truncated power spline which interpolates the data - a form of which was described in “3.3.2 Truncated Power Splines in Multiple Dimensions” on page 51.

MARS uses GCV as a criterion to choose the best partition at each iteration. Any of the regions, constructed from the previous iteration, can be partitioned along any of the parameters with a knot at any of the parameter values belonging to the region. In theory, any data point can serve as a knot, but in practice, to avoid biasing segments of the model by a run of observations which are consecutively either below or above their mean, a minimum span of observations between each knot is required. MARS examines all the possible partitions by constructing a set of basis functions for each one and performing

regression over each set. The partition resulting in the lowest GCV is chosen and the basis functions representing that partition are added to the model.

The procedure just outlined is referred to as the “forward stepwise phase” and it finishes once a user-specified, maximum number of functions has been added to the model. The forward stepwise phase, however, introduces some problems. For one, adding complementary pairs of truncated power functions to the model can render some of the functions redundant. In a 1-parameter situation, if the data has been split at points x_{k1} and x_{k2} , $x_{k1} < x_{k2}$, the model will contain the following basis functions: $(x-x_{k1})_+$, $-(x-x_{k1}))_+$, $(x-x_{k2})_+$, and $-(x-x_{k2}))_+$. With only two knots and three regions, only three basis functions are required - so either $-(x-x_{k2}))_+$ or $(x-x_{k1})_+$ can be dropped. Secondly, as basis functions are added to the model, there will come a point when the next two basis functions chosen do not reduce the average squared residual substantially enough to offset the cost incurred by using the extra degrees of freedom. Consequently, as the model is built its GCV will decrease up to a point and will then begin to rise again. In response to this situation, after completing the forward stepwise phase, MARS begins a backward stepwise phase in which it examines each basis function and decides if it can be removed from the function. The criteria for removing a basis function is, again, GCV. A basis

function is removed, regression is performed over the remaining function, and the GCV is calculated. The result of the backward stepwise process will be a set of basis functions which result in the minimum GCV. This model is the final product of a MARS run.

3.5.1 Degrees of Freedom and the Maximum Number of Basis Functions

MARS employs GCV to measure how well a particular model fits the data. If you recall, GCV is defined as:

$$GCV = \frac{1}{N} \cdot \frac{\sum_{i=1}^N (y_{oi} - f_{est}(X_i))^2}{\left(1 - \frac{df}{N}\right)^2}$$

Since the position of knots is chosen adaptively from the data, fitting each basis function consumes more than one degree of freedom per function. It is not clear, however, what the exact value of df should be. Friedman employs the following formula as a quick, simplistic way of estimating the number of degrees of freedom used up in a fit:

$$df = B + s(B - 1)$$

where B is the total number of basis functions in the model. Since there is only 1 constant function in the model, $B-1$ represents the number of non-constant basis functions. The parameter s is referred to as the “smoother” and it represents the additional cost in degrees

of freedom for each non-constant basis function. The smoother is user-specified although Friedman suggests using values between 2 and 3. A smaller value of s will favour models with lower residuals and a tighter fit. A larger value of s is applicable in situations with higher measurement error, where a less volatile model is desired. In the low noise validation tests conducted in Chapter 5, we used a value of 2 for the smoother, whereas in the higher noise case study in Chapter 6, the smoother was assigned a value of 3.

Given a value of s , we can determine the maximum number of basis functions which can be considered for a model:

$$N > B + s \times (B - 1)$$
$$\frac{N + s}{1 + s} > B$$

This follows from the fact that the number of degrees of freedom of a model cannot exceed the number of points on which the model is based.

A Method for Limiting Detail

4.0 Overview

This chapter presents a practical method for limiting the amount of detail required in the construction of a resource function. By detail, we mean the number and location of design points (points at which the resource demand is measured). To limit the detail, the method incrementally adds design points to the experiment, models the measurements using MARS, and stops when the MARS model achieves the required accuracy. Here, the accuracy of the model is defined as the percentage difference between the MARS function and the “true” function. Since we do not know what the true function is, a GCV-based pseudo-confidence interval around the MARS model is used to delimit where the true function could lie. The construction process stops when the pseudo-

confidence interval is small enough. This chapter begins with a discussion on how the user might specify a desired accuracy by relating it to the sensitivity of the performance model in which the resource demand estimates will be used. Section 2 then discusses how to estimate the accuracy of the MARS model. Section 3 describes how the accuracy of a MARS model can improve by adding more points. Section 4 describes a strategy for choosing the next points to add to the model. Section 5 outlines the whole technique for incrementally building a resource function.

4.1 Choosing the Accuracy

The impact of resource function accuracy on performance model prediction has already been described in Chapter 2. The elasticity of a performance model's response with respect to the performance model's parameters can be used to set the required accuracy of the resource functions for each of those parameters. With the research done by Sheikh [Sheikh97], point elasticities can in principle be calculated as part of the model solution process. To build a resource function for a specific performance model, then, requires that the modeler engage in an iterative process. The resource function would have to be constructed initially to conform to an arbitrary, reasonable accuracy. The layered model would then be created based on the parameter estimates provided by the

resource functions. As part of the solving process, the sensitivities of each response to each model parameter would be extracted, and based on these sensitivities, point elasticities could be calculated. The elasticities along with the initial accuracy of the resource function could be used to calculate the percentage error of the model's responses. The modeler would then decide if the original resource function was "good enough" or if the accuracy requirements needed to be tightened. If greater accuracy was required, the resource function would have to be upgraded to satisfy the new requirements. In the case of prepackaged resource functions, the resource function builder cannot know in advance the accuracy requirements of every user of that function. An arbitrary, reasonable limit on the function's accuracy, therefore, would have to be imposed. Expert opinion could be relied on to provide some guidelines. Suppose for a particular performance model, a 20% error in the model's prediction is tolerable. Assuming the elasticity of a model response with respect to any given parameter is less than 1, a resource function accuracy between 10% and 20% would be a "reasonable" value in such a case. Throughout the validation tests (Chapter 5) and the case study (Chapter 6) an accuracy requirement of 10% was imposed on the resource functions.

4.2 Estimating the Accuracy of a MARS Model

We define the accuracy of a MARS model as the largest percentage deviation of the model from the “true” function. Since the true function is unknown, a pseudo-confidence interval is used to estimate how far the MARS model deviates from the “true” functions. Chapter 3 described two techniques for calculating a confidence interval around a regression model: the standard MSE-based approach and the bootstrap method. The computationally expensive bootstrap method is not a practical option for us because our construction technique is incremental and the accuracy of the model must be tested frequently. Instead, we calculate the variance of the model estimates with a formula similar to the MSE-based approach, replacing MSE with GCV.

4.2.1 GCV-Based Pseudo-Confidence Intervals

When modeling measurement data, the assumptions of normally distributed, homoscedastic residuals may not hold. As such an MSE-based approach can underestimate the correct confidence interval width. In the case of MARS modeling, though, as more design points are added, the fit actually improves (discussed later), thus bringing the distribution of error closer to normality and making the MSE-based approach more realistic. It is hoped that using GCV instead of MSE will widen the confidence

interval enough to compensate in cases where the assumptions are not met. The “true” function, then, is assumed to fall within 3 standard deviations from the MARS model. i.e.

$$\pm 3 \cdot \sqrt{GCV(tv(X_i) \cdot (TM^T TM)^{-1} \cdot tv'(X_i))}$$

If the residuals were normally distributed and MSE was used in place of GCV, this calculation would give a confidence level above 99% for the interval.

In our technique, this pseudo-confidence interval is not calculated on the raw observations. Since the accuracy of our model is based on the notion of percentage difference, we first transform the observed data and model estimates into the log domain before calculating the GCV-based pseudo-confidence intervals. In this way, observations which deviate from the model by the same percentage receive the same residual value. Having calculated the interval in the log domain, it is then mapped back to the real domain to calculate the percentage difference between the MARS model and the true function. An example of a confidence band around a model is shown in Figure 4 on page 64.

FIGURE 4. Model and Confidence Interval in the Log Domain

Let d , be the distance between the edge of the pseudo-confidence interval and the model at point X_i , and let l be the log-transformed value of the MARS model at point X_i . Assuming the true function lies on the boundary of the pseudo-confidence interval, the percentage difference between the true function and the model in the real domain is:

$$\frac{e^{l+d} - e^l}{e^l}$$

It is easy to show that the percentage difference between the true function and the MARS model at a point X_i is :

$$e^{\frac{3 \cdot \sqrt{GCV_{\log}(tv(X_i) \cdot (TM^*TM)^{-1} \cdot tv(X_i))}}{2}} - 1$$

where $GCV_{\log}$ is calculated as:

$$GCV_{\log} = \frac{1}{N} \cdot \frac{\sum_{i=1}^N (\log(y_{o_i}) - \log(f_{est}(X_i)))^2}{\left(1 - \frac{df}{N}\right)^2}$$

The width of the confidence interval and the corresponding percentage deviation between the true function and the model will vary for each X_i . As an estimate for the accuracy, call it EA , we choose the largest percentage difference. That is:

$$EA = \max_{i=1, \dots, N} (e^{3 \cdot \sqrt{GCV_{\log}(iv(X_i)) \cdot (TMTM)^{-1} \cdot iv(X_i)}} - 1)$$

The interval at each point is referred to as a pseudo-confidence interval because it is based on GCV and an exact level of confidence cannot be quoted. In the validation experiments of Chapter 5, however, the probability of the MARS model deviating by no more than EA percent from the “true” function is shown to be at least 0.9, provided the accuracy estimate is used in conjunction with the point selection algorithm to be described shortly.

4.3 Improving the Accuracy of a MARS Model

The fit of a MARS model becomes more accurate as more points are added to the design. To see this, examine the denominator in the GCV calculation. As the number of points increases, the cost of adding an extra basis function to the model decreases. And the more basis functions in the model, the better the fit. Once the fit is close to the underlying function, the ASR for the current model will remain the same as more points are added. The denominator of GCV, however, will increase, resulting in a decreased GCV value, a tighter pseudo-confidence interval and, therefore, a better accuracy.

A simple example will illustrate this point. The following figures demonstrate how the MARS fit improves as points are added to a design. The underlying function is defined as:

$$f(x) = \begin{cases} 100 \sin(x) & 0 \leq x \leq 90 \\ x - 50 & x > 90 \end{cases}$$

4.3 Improving the Accuracy of a MARS Model

FIGURE 5. Increasing the accuracy of MARS, N=5 points

FIGURE 6. N=10 points

4.3 Improving the Accuracy of a MARS Model

FIGURE 7. $N=20$ points

FIGURE 8. $N=40$ points

The fit becomes better with more points because additional points allow MARS to use more basis functions in fitting the data. Consider two candidate MARS models for fitting

N data points: Model A and Model B. Model A uses b basis functions with an average squared residual, ASR_A . Model B uses $b+1$ basis functions with an average squared residual, ASR_B . The model which is chosen by the MARS procedure will have the smaller GCV. So consider what would be required if Model B were to be chosen over Model A:

$$\begin{aligned}
 GCV_B &< GCV_A \\
 \frac{ASR_B}{\left(1 - \frac{(b+1 + s \cdot (b-1+1))}{N}\right)^2} &< \frac{ASR_A}{\left(1 - \frac{(b + s \cdot (b-1))}{N}\right)^2} \\
 ASR_B &< ASR_A \cdot \left(\frac{N-b-1-(s \cdot b) + s-s}{N-b-(s \cdot b) + s}\right)^2 \\
 ASR_B &< ASR_A \cdot \left(1 - \frac{(s+1)}{N-b-(s \cdot b) + s}\right)^2 \\
 ASR_B &< ASR_A \cdot \left(1 - \frac{(s+1)}{N-df_A}\right)^2
 \end{aligned}$$

where df_A is the degrees of freedom used up by model A and s is the smoother. It is obvious that ASR_B must be smaller than ASR_A in order for Model B to be chosen.

However, as N increases, the percentage by which ASR_B has to be smaller decreases. And so it becomes increasingly more likely to choose models with more basis functions as more points are added to the model - and more basis functions permit a closer fit.

As more points are added to the experimental design, the MARS model will start to approach the shape of the underlying “true” function. When a particular model is close to the “true” function, adding more design points will not change its ASR, call it ASR_c . Therefore, if Δ new points are added, the GCV of that particular model, GCV_c , will drop to GCV_c^Δ i.e.

$$GCV_c = \frac{ASR_c}{\left(1 - \frac{df_c}{N}\right)^2} > \frac{ASR_c}{\left(1 - \frac{df_c}{N + \Delta}\right)^2} = GCV_c^\Delta$$

When the new points are added and the MARS algorithm is run, that particular model is, of course, only a candidate among possible models which will be chosen. Since the MARS algorithm will choose the model with the minimum GCV, the GCV of the next model based on the additional data will be less than or equal to the GCV of the current model based on the additional data. That is $GCV_c > GCV_c^\Delta \geq GCV_{next}$. And as the GCV score decreases, the confidence interval will tighten resulting in a smaller percentage error.

All that is needed now is a strategy for selecting new points to add to the model if the confidence band around the current resource function is not tight enough to satisfy the error requirements of the performance model.

4.4 A Strategy for Selecting Additional Points

One goal in selecting new points is to have them evenly distributed throughout the design space. In this way the measurements give equal attention to all areas in the parameter domain. It is also desirable to place new points near areas of high deviation (from the current model) in order to confirm that the high deviation is caused by outliers or, if that is not the case, to reinforce the behaviour in that region. Reinforcing behaviour in regions of high deviation increases the average squared residual of the current model and so encourages MARS to opt for a model which more closely fits that region.

Given these two goals, a metric has been developed which assigns a value to each point in the design space. This metric is referred to as the Average Repelling Force. Points which receive a lower value of ARF are better candidates to add to the model. The idea behind the metric is as follows:

- Each existing design point in the current model exerts a repelling force on new points which diminishes with distance
- Existing design points at which there is a larger percentage deviation between the current MARS model and the observed value exert a softer force than those design points at which there is lower percentage deviation.
- The best new design points will be those points in the design space which experience the lowest average repelling force.

Here, distance is defined as the unit euclidean distance. Suppose there are two points, X_i and X_j , in K -space. That is $X_i = [x1_i, x2_i, \dots, xK_i]$. The unit euclidean distance between them is calculated as follows:

$$\|X_i - X_j\|_{unit} = \sqrt{\sum_{p=1}^K \left(\frac{x p_i - x p_j}{\max_{a=1 \dots N}(x p_a) - \min_{a=1 \dots N}(x p_a)} \right)^2}$$

The difference between the values of each independent parameter, p , is divided by the total width of that parameter's domain i.e. $\max_{a=1 \dots N}(x p_a) - \min_{a=1 \dots N}(x p_a)$. In this way, the value of each parameter is implicitly transformed into a value between 0 and 1. This modified distance is used because, in a particular experiment, the domain of each

parameter may differ by orders of magnitude. Using this transformation prevents one parameter from completely dominating the distance calculation.

To encourage placement near areas of high deviation, we would like the repelling force exerted by an existing point to be softened by an amount proportional to its deviation from the model. The deviation of a current point is defined as its percentage deviation from the existing model. Suppose there are two existing design points X_i and X_j , where the observation at X_i deviates from the current model by $p*100$ percent and the observation at X_j deviates by $2p$. (Here, we don't consider the sign, so p is never negative). If there is a new point, X_{new} , equally distant from X_i and X_j , then we would like the repelling force exerted on X_{new} by X_j to be half as much as that exerted by X_i .

A problem arises, though, if one of the existing points happens to have zero deviation since the deviations of the other points will be infinitely times greater. In such a case, if an existing point, X_i , has zero deviation, the repelling force it exerts on X_{new} is simply:

$$RF(X_i, X_{new}) = \frac{1}{\|X_i - X_{new}\|_{unit}}$$

Otherwise, if the observation at X_i deviates from the model by $PercentDev_i * 100$ percent, the repelling force which it exerts on X_{new} is:

$$RF(X_i, X_{new}) = \frac{1}{\|X_i - X_{new}\|_{unit} \cdot (1 + minPercentDev)} \cdot \frac{minPercentDev}{PercentDev_i}$$

where $minPercentDev$ is the minimum non-zero percent deviation over all the existing points. $PercentDev_i$ and $minPercentDev$ are, again, unsigned and always positive. In this way, the point with the minimum non-zero deviation exerts a force of $1/(1 + minPercentDev)$ times that of a the same point with zero deviation. All other points have their force reduced by how many times their deviation is greater than the minimum non-zero deviation. So the higher the deviation, the more quickly the repelling force diminishes with distance.

The average repelling force on a potential point, X_{new} , is the average of the repelling forces from each of the existing points in the neighbourhood of X_{new} . That is:

$$ARF(X_{new}) = \frac{\sum_{i \in Neighbourhood(X_{new})} RF(X_i, X_{new})}{|Neighbourhood(X_{new})|}$$

If you imagine a set of axes centered on the new, candidate point, its neighbourhood is defined as including the closest existing point from each quadrant formed by the axes. Using a neighbourhood allows the algorithm to assign lower values of the metric to points in emptier spaces in the experimental design. The reason for not basing ARF on the repelling force of all points is that such a strategy tends to favour points at the extreme ends of the design space. Consider the case where there is only one independent parameter with four existing design points (a,b,c,d) and three candidate points (A, B, C) in between each existing point. Graphically, it might look like:

The numbers represent the distances between points. Assuming the existing points have zero deviation from the model, the ARF based on all points is .044868 for A and C but .044961 for B, even though B is located in a emptier space. Basing ARF on neighbourhood, however, gives B a value of .067 (from b and c) whereas A and C receive values of .071.

In this way, points which are in empty spaces and, in particular, empty spaces adjacent to regions of high deviation are candidates to add to the model. Points are randomly chosen throughout the parameter domain and the average repelling force is calculated for each of them. By randomly selecting many points and calculating their ARF, the outline of the ARF function can be discovered and the point which approximates the minima can be chosen. As an example, the model and points from Figure 6 on page 67 result in the following ARF curve calculated for 1000 randomly chosen points:

FIGURE 9. ARF of 1000 Randomly Chosen Points

Singularities occur at the current design points since a new point at the exact location of an existing point makes the denominator in the repelling force calculation equal to zero.

Furthermore, the new points whose neighbours have high deviation receive a lower ARF. The best candidates, in this case, are in between 80 and 120.

4.5 An Incremental Resource Function Construction Technique

Given a set of K independent parameters, and a range of values for each parameter representing the user's region of interest, the method proposed in this thesis begins with a 2^K factorial experimental design plus a set of points scattered randomly throughout the design space. The factorial points provide a foundation for the design and delimit its boundaries. The number of randomly scattered points is calculated as $K(2 + 2s)$ thus allowing MARS to partition the points up to K times.

After the first set of points are collected, a MARS model is built. Since the accuracy of the model is based on percentage deviation, it is desirable to fit the smaller observations more closely than the larger ones. To accomplish this, a set of weights is provided for the regression which MARS performs over the selected basis functions. If the observation at a point is yo_i , then the weight given to that point is $1/\log(yo_i)$, normalized so that the sum of all weights is 1. The accuracy of the MARS model is then estimated using the widest percentage-wise section of the GCV-based confidence band.

If the accuracy does not satisfy the requirement, a new point is chosen to add to the design. To find this new point, a batch of points are randomly generated, the ARF value for each is calculated, and the point with the minimum ARF is selected. The size of this batch should be large in comparison to the existing number of points. The batch size chosen in the implementation (described in the next chapter) was $1000 * K$, where K is the number of independent parameters. Although this is a large number of points, in comparison to the time required to collect a measurement and model the data with MARS, experience has shown that the next point calculation does not impose much overhead. The advantage to generating a large number of points is that it improves the chances of selecting the optimal point.

The process of selecting a new point, performing the measurement, modeling the data and estimating the accuracy is continued until the pseudo-confidence interval is tight enough to satisfy the accuracy requirement or until the number of design points reaches a user-specified limit.

Implementation and Validation

5.0 Overview

The technique described in the previous chapter was implemented as a Tcl script, called “smartexec”, and was integrated into the Performance Workbench[Vetland96]. This new script serves as an alternative to Vetland’s “executor” which could only perform factorial experiments. The first section of this chapter provides a description of the implementation. The second section presents the validation tests which were performed to verify whether *EA* (Chapter 4) provided a realistic estimate of accuracy and whether the overall technique was able to produce resource functions satisfying the accuracy requirements of the user.

5.1 An Extension to the Performance Workbench

5.1.1 The Experiment Launcher

The original workbench was designed to facilitate factorial experiments. When an experiment was to be launched, the user was provided with the following interface:

New Measurement Experiment

Clear Run Quit

Harness program funharn

Machine soyuz
sunspot
suntan

Parameters

The values of x could be specified as: *from:to:step* where *from* was the start of the domain, *to* was the end of the domain and *step* was the desired interval between each parameter value in the experiment. If other parameters were specified, all of their desired values

could be specified in the same way, and the “executor” script would construct a complete factorial experiment out of those values. With the extensions made to the workbench, the experiment launch window now looks like:

New Measurement Experiment

Clear Run Quit

Factorial Design Incremental Design

Description: Simulated TCP/IP Send

Harness program: funharn

Machine: soyuz
sunspot
suntan

Parameters

SMART OPTIONS

Response Focus Variable: CPU

Smoother: 2

Accuracy: .1

Maximum Number of Points: 80

MIN: 1

MAX: 5000

GRANULARITY: 1

The two radio buttons at the top allow the user to choose between a normal factorial experiment (as before) or a MARS-based incremental experiment. If the incremental process is selected, some additional information will have to be provided in the “SMART OPTIONS” sub-window. For one, since a harness can return values for more than one result variable at a given design point, the modeler must specify which of these variables the incremental process should focus on (that is, which MARS model should be used to determine if the accuracy is adequate). Since additional design points are chosen based on the distribution of the existing points in addition to the closeness of fit of the MARS model, the process must be told which model to use when looking at residual percentage deviations. Since most experiments are designed to return only one result variable, this is usually not an issue.

The smoother is the additional cost, in degrees of freedom, incurred by non-constant basis functions. As described in Chapter 3, this variable usually assumes values between 2 (for low noise data) and 3 (for high noise data). The accuracy specifies how close a fit the experimenter desires. The accuracy of the resource function, as defined in chapter 4, is compared against this user-provided value to determine if the experiment should stop or continue with additional design points. The user must also specify the

maximum number of points to collect during an experiment. Even if the accuracy is not acquired by the time this number of points is collected, the process will stop.

For each independent parameter in the experiment, the user must provided the starting and ending value of its domain. These correspond to the MIN and MAX entries on the window. The granularity restricts what values can be chosen by the incremental procedure for experiments. A 0 specifies that all real numbers within the domain are eligible candidates. A 1 means that every integer is a candidate. A 2 would mean that every second integer was a candidate. A .5 would indicate that any number in the domain which is a multiple of .5 is a possibility.

5.1.2 Smartexec

If the user has chosen an incremental experiment, when the “Run” button is hit, the “smartexec” script will be invoked. The script generates an original set of design points corresponding to a 2^K factorial experiment with an additional $K(2+2s)$ points scattered randomly through the design space (Chapter 4). Once a set of design points has been measured, an input file is constructed and a program is invoked which performs the MARS modeling, estimates the accuracy and selects the next point. When it completes,

the program produces an output file containing the MARS model, statistics used for verifying the stopping condition, and the best design point to use next should the script decide that more iterations are required. The overall structure looks like:

5.1.2.1 Format of the Input File

```
<number of data files>
<name of data file 1>
<name of data file 2>
...
<number of columns in data files>
<number of predictors>
<predictor 1 name> <column number> <start domain> <end domain> <granularity>
<predictor 2 name> <column number> <start domain> <end domain> <granularity>
...
<name of focus result variable> <column number of focus result variable>
<maximum number of basis functions>
<smoother>
```

Note, the $\langle \rangle$ simply means that the contents are specified by the calling process. The actual characters “<” and “>” should not appear in the file.

Recall that the smartexec script generates an original set of design points and performs an experiment on those points. The results of those experiments and the corresponding values of the independent parameters are stored in the Performance Workbench Database. This database consists simply of one header file and a number of flat files containing the raw data from each experiment. Once the first iteration is performed, a new data file will exist in the database directory. It is this file which the MARS modeling will be run against and as such its name must be specified in the input file. Multiple data files are permitted, provided they have the same number of columns and each column has the same meaning.

For each predictor variable, the input file must contain the name of the variable along with its column location in the database file. Since the MARS program is also responsible for generating the next batch of candidate design points, the domain and granularity of each predictor must be provided. Furthermore, since multiple result variables can be measured in one experiment run, the modeling process must be provided with the column number of the result variable to focus on.

Finally, the last two rows of the input file contain the value of the maximum number of basis function and the MARS model smoother. If the maximum number of basis functions is set to -1, the system will choose the maximum number of basis functions based on the smoother value and the number of points.

5.1.2.2 Format of the Output File

```
MARS MODEL
<model>
GCV
<gcv>
DF
<degrees of freedom>
TOTAL POINTS
<total number of points>
PERCENT ACCURACY
<percent accuracy>
NEXT POINT
<next design point>
MARS vs REAL
<current design point 1> <MARS response estimate> <Observed response> <percent diff>
<current design point 2> <MARS response estimate> <Observed response> <percent diff>
...
```

The model returned is in a compressed format which can be easily parsed. Additional executables have been built to parse the model and display it in a human-readable format. GCV is the model's GCV calculated by MARS. DF is the number of degrees of freedom used up by the fit. TOTAL POINTS is simply the number of points on which the fit was based. PERCENT ACCURACY is largest width of the GCV-based confidence interval in percentage terms (Chapter 4). The NEXT POINT section specifies the design point to measure next. It is found by generating $1000 \cdot K$ points and selecting the point with the minimum ARF. The MARS vs REAL section simply provides a comparison of the MARS estimate and the observed response at each design point.

5.1.3 The Resource Function Manager

The Resource Function Manager is a feature which has been added to the Performance Workbench for the purpose of storing, displaying and evaluating MARS resource functions. When the incremental construction process is complete, smartexec invokes the resource function manager to store the final MARS model. This model will be stored in a resource function file within the workbench's database directory. Having stored a resource function with the resource function manager, it is available for viewing by the user. The following figure is a snapshot of the resource function browser window:

The selected resource function is displayed in a “prettified” format and the user can evaluate the function for given values of the independent parameters.

5.1.4 Choosing the Next Design Point

Apart from the MARS algorithm, the most computationally expensive aspect to the procedure is calculating the best design point to use next. The algorithm used in this implementation is of order $O(M(N+N^2))$ where M is the number of points randomly generated and N is the number of existing points in the design. For each candidate point, the algorithm scans through the existing set of design points to find the closest point in each quadrant formed by the set of axes centered on the candidate point. An existing point, however, may fall directly on one of these axes and so a mechanism is required to determine which quadrant the point should be assigned to. For an existing point, each of its independent parameter values may be less than, equal to, or greater than the corresponding parameter value for the candidate point. As such, a category can be constructed for each combination of parameter value locations with respect to the candidate point's parameter values. There are 3^K such categories (K is the number of independent parameters) but no more than N can possibly be referenced if 3^K happens to be greater than N . The idea then is to scan the existing points once to determine the

closest point in each category. The closest points from each category are then sorted in ascending order according to proximity to the candidate point. Beginning with the closest point, the algorithm steps through the sorted list, until each quadrant is uniquely represented by one point.

5.1.5 Complexity

It has already been stated that the complexity of the point selection algorithm is $O(M(N+N^2))$. The complexity of the MARS algorithm, on the other hand, is roughly $O(KN(B_{max}^3+B_{max}^4))$ where K is the number of independent parameters, N is the number of points in the design and B_{max} is the maximum number of basis functions permitted in the MARS model. Since the computation time of MARS is very sensitive to the maximum number of basis functions, Friedman suggests using no more than 50 basis functions. The point selection algorithm, however, is more sensitive to the number of design points and begins to slow down the construction process noticeably when more than 1000 design points are involved. Clearly then, the technique proposed in this thesis has limits both with respect to the type of function which can be fit and the number of

measurements which can be collected. For complex resource functions where more than 50 basis functions and/or more than 1000 measurements are required to achieve the desired accuracy, the incremental technique becomes impractical.

5.2 Validation

Two types of validation tests were performed. The purpose of the first test was to see how closely the estimated accuracy, *EA*, represents the actual percentage deviation between a MARS model and the true function when it is used in conjunction with the point selection algorithm described in Chapter 4 . The purpose of the second was to test how well the incremental resource function construction process works when applied to a variety of known functions.

5.2.1 Validating *EA*

EA estimates the percentage deviation between the MARS model and the true function. As described in section 4.2.1, *EA* is a heuristic estimate based on the pseudo-confidence band around a MARS model. The heuristic was validated by sampling points (with noise) from a known function, modeling the points with MARS, calculating the *EA* for that model and then determining the real percentage deviation. For our tests, 30

functions were randomly generated. For a given function, an initial set of data points was collected and modeled with MARS. The initial set of design points was determined in the same way as it is in the resource function construction technique: that is, 1 point at each extreme of the independent parameter's domain plus $(2+2s)$ randomly scattered points. In all cases a value of 2 was used for the smoother. Using the point selection algorithm, points were then chosen from the function until 1000 points were collected. After each point was collected, the data was modeled with MARS, the next point to measure was selected, and the *EA* for the model was calculated. At intervals of 25 points up to the first 200 points, the *EA* of the model was compared to the actual percentage error. From then on, intervals of 100 points were used.

To find the actual percentage error, 10000 points were randomly sampled from the domain of the independent parameter, and the value of the true function at those points was compared to the estimate from the MARS model. The percentage errors at each point were ranked to determine the 100th, 99th, 95th, and 90th percentile. Let PE_p denote the p th percentile of percentage errors i.e. p percent of the percentage errors are less than or equal to PE_p . The *EA* was then compared to these percentiles.

5.2.1.1 Randomly Generating the Functions

Truncated power splines with one independent variable were used to generate the functions. Each spline was comprised of 100 linear segments. The length of each segment was a randomly chosen number between 0 and 10. The coefficient of each term in the spline was randomly chosen to be between -100 and +100. A large number of small segments were used to approximate non-linear functions. A constant was then added to ensure that the values of the true function were all positive. In all cases, the MARS models (which were used to fit the data sampled from these functions) were restricted to a maximum of 30 basis functions in order to test the effectiveness of *EA* when a perfect fit is impossible.

5.2.1.2 Noise

A normally distributed measurement error was added to each observation so that 99% of the observations fell within 10% of the true value. Notice that in doing this, the errors are no longer homoscedastic since the variance grows with larger observations. In this way we can see how good our GCV-based accuracy estimate is when the assumptions required for a valid MSE-based confidence interval do not hold.

5.2.1.3 In Conjunction with the Point Selection Algorithm

The purpose of the validation test was not to determine how well *EA* works in general, but how well it works in the context of the incremental resource function construction process. As such, the points were chosen using the same point selection algorithm as the one used in the construction technique. Since the selection algorithm places points in the areas which need the most modeling attention, the points (especially with lower sample sizes) are more likely to represent the peculiarities of a given function.

5.2.1.4 Goal

The results from this validation test determine what the probability is that the percentage error of a MARS model at a given design point, $PercentageError(f_{est}(X_i))$, will lie within *EA* - that is:

$$Prob(PercentageError(f_{est}(X_i)) \leq EA)$$

Ideally we would like this probability to be high - .99 for instance. In such a case, we would expect that each *EA* calculated in the validation test would be greater than or equal to the 99th percentile, i.e. $EA \geq PE_{99}$. Since *EA* is just a heuristic, however, we will consider it an adequate estimate of accuracy if $EA \geq PE_{90}$ for all trials.

5.2.1.5 Results

TABLE 1. Correctness of *EA*: 30 Generated Functions, Intervals of 25 & 100 points

# points in model when <i>EA</i> calculated	# of times $EA \geq PE_{100}$	# of times $PE_{100} > EA \geq PE_{99}$	# of times $PE_{99} > EA \geq PE_{95}$	# of times $PE_{95} > EA \geq PE_{90}$	# of times $PE_{90} > EA$
25	19	10	1	0	0
50	15	12	3	0	0
75	5	21	4	0	0
100	5	21	4	0	0
125	2	20	8	0	0
150	3	15	11	1	0
175	2	19	8	1	0
200	1	14	14	1	0
300	1	15	13	1	0
400	0	10	18	2	0
500	0	7	20	3	0
600	0	6	22	2	0
700	0	8	17	5	0
800	3	8	13	6	0
900	0	7	17	6	0
1000	0	6	15	9	0

EA is often greater than the 100th percentile when a small number of observations is involved. As more points are added, however, it seems that the pseudo-confidence interval shrinks more quickly than the “real” confidence interval. Since the MARS model will never perfectly match the underlying real function, bias will never be completely eliminated. And since confidence intervals, even the expanded pseudo-confidence intervals, implicitly assume that all error is from measurement error, there will come a

point (after adding a certain number of observations to the model) when even a small bias in the MARS model will render the model less accurate than the confidence intervals claim. Nevertheless, this phenomenon only seems to occur at high levels of accuracy. Figure 10, “Correctness of EA,” on page 96 shows, for each EA calculated, the percentile PE_p such that $PE_p \leq EA < PE_{p+1}$ versus the actual percentage deviation of the 90th percentile.

FIGURE 10. Correctness of EA

What the plot shows, is that EA only drifts into the 90th percentile region when the percentage deviation of the 90th percentile dips below 10%. Above that, it seems to approximate the 95th to 99th percentile of percentage deviations between the MARS

model and the true function. In the worst case, therefore, the results indicate that EA approximates the 90th percentile when up to 1000 observations are involved. That is:

$$Prob(PercentageError(f_{est}(X_i)) \leq EA) \geq 0.9$$

5.2.2 Validating the Construction Process

In order to validate the resource function construction process, it was run against four types of functions to see how closely the resulting MARS models came to the “true” models. These four types were: a straight line, a log function, a second order polynomial, and an approximation to Pozzetti et al’s discontinuous, non-linear signature for TCP/IP send between two HP machines. In all cases, the desired accuracy was set to 10% and the smoother was 2. For the first three, x values ranged from 1 to 200 (the range is arbitrary in this case) and the granularity was set to 0. For Pozzetti et al’s signature, the granularity was 1 (because you can’t send less than 1 byte)and the message size ranged from 1 to 5000 bytes because that was the range used by Pozzetti et al. Once again, a random, normally distributed error was added so that 99% of the observations fell within 10% of the true value. For each of the MARS models generated, two histograms are provided - one to indicate percentage residual error between the observations and the model, and another to indicate the percentage difference between values of the true function at 1000

randomly selected points and their estimates provided by the model. The goal of the validation is to determine how well MARS models the data, how close the estimated accuracy reflects the actual accuracy and, how many points are required in each experiment. The following sections present the results:

5.2.2.1 A Linear Function

FIGURE 11. Linear Function: $y = 3x + 100$

- Required Number of Points: 13
- Estimated Accuracy (EA): 8.28%
- MARS Model: $y = 102.896 + 3.09692[+(x-1)]_+$
or $y = 99.799 + 3.09692x, \quad x \geq 1$

FIGURE 12. Residuals as a Percentage of the Observed Values

FIGURE 13. Percentage Difference between 1000 True Points and the Model:

We would expect for MARS to have no trouble in fitting a straight line. The interesting feature here is that the GCV-based accuracy estimate of 8.284% is very large since, in the worst case, the MARS model deviates from the true function by no more than 3 %.

5.2.2.2 A Log Function

FIGURE 14. Log Function: $y = 10\ln(x) + 10$

- Required Number of Points: 54
- Estimated Accuracy (EA): 8.32%

$$\begin{aligned} \text{MARS Model: } y = & 83.293 + 4.2309[+(x-17.0605)]_+ - 4.5216[-(x-17.0605)]_+ \\ & - 3.72661[+(x-4.9257)]_+ - 0.1372[+(x-143.785)]_+ \\ & - 0.3634[+(x-33.4954)]_+ \end{aligned}$$

FIGURE 15. Residuals as a Percentage of the Observed Values

FIGURE 16. Percentage Difference between 1000 True Points and the Model:

In this case, the non-linear nature of the underlying function was responsible for the additional points required to fit it. Even with heteroscedastic residuals and increased bias, EA is greater than the 99th percentile of percentage deviations between the MARS model and the true function at the 1000 randomly selected points. That is, 99% of the MARS model estimates fall within 6.1 % of the true function whereas the estimated accuracy was 8.32%. Thus the accuracy estimate, based on the pseudo-confidence interval, is, in this case, slightly conservative. An examination of the design point distribution shows that the point selection algorithm favored points in the region where the curve is sharpest - between 0 and 40. This occurs because the sharper a curve is, the more difficult it is to model with linear segments. In such an area, the percentage deviation of observations from the model, therefore, is likely to be higher than in the rest of the design space.

5.2.2.3 A 2nd Order Polynomial

FIGURE 17. 2nd Order Polynomial: $y = 2x^2 + 5x + 1000$

- Required Number of Points: 92
- Estimated Accuracy (EA): 8.16%

$$\begin{aligned}
 \text{MARS Model: } y = & 4677.65 + 206.35[+(x-92.2997)]_+ - 40.9109[-(x-92.2997)]_+ \\
 & + 264.798[+(x-140.618)]_+ + 97.1628[+(x-19.4755)]_+ \\
 & + 156.676[+(x-53.2253)]_+
 \end{aligned}$$

FIGURE 18. Residuals as a Percentage of the Observed Values

FIGURE 19. Percentage Difference between 1000 True Points and the Model:

Again, EA is greater than the 99th percentile of percentage deviations. That is, 99% of the MARS estimates are within 7.88% of the true points while the estimated accuracy was 8.16%. The distribution of design points also shows the effect of the point selection algorithm. Since the observations span several orders of magnitude, it is likely that the highest percentage error will be found near the smaller-valued observations. In order to ensure that MARS fits this region more closely, more design points are concentrated there.

5.2.2.4 A discontinuous function

For completeness, the incremental construction process was run against an approximation of Pozzetti et al's TCP/IP Send "signature" between two HPUX machines, shown in Chapter 1. It was mentioned in Chapter 3, that MARS imposed a minimum span of observations between each knot. This strategy presents a problem when modeling discontinuous functions, since at the break point, the two consecutive observations on each side of the discontinuity should be used as knots. When measurements were sampled from the simulated signature, Friedman's MARS implementation had difficulty fitting the points near the discontinuities. To overcome this problem, the MARS algorithm was implemented in C++ with no restriction on the minimum number of observations between knots. This new version provided a much better fit. In this section, we first present the fit provided by Friedman's implementation of MARS. Following that, the fit calculated by the new implementation is demonstrated. Since Friedman's implementation was not able to achieve the required accuracy of 10%, we present Friedman's fit on the same observations that satisfied the accuracy requirement when the new implementation was used. The histograms of residual percentage deviations and percentage deviations from the true function are only presented for the new implementation. In both cases, the function used to simulate the signature was:

$$y = \begin{cases} 0.55x + 100 & x < 1000 \\ 0.05x + 250 & 1000 \leq x < 2000 \\ 0.75x - 1150 & 2000 \leq x < 3000 \\ 0.1x + 400 & 3000 \leq x < 4000 \\ 0.7x - 2000 & 4000 \leq x < 5000 \end{cases}$$

FIGURE 20. Friedman's Fit of Pozzetti et al's "Signature"

- Number of Points Used (same used for new implementation): 102
- Estimated Accuracy (*EA*): 20.8%
- MARS Model: $y = 2328.58 + 1.1093[+(x-4076)]_+ - 0.5457[-(x-4076)]_+ - 2.7112[+(x-2995)]_+ + 0.7613[+(x-1985)]_+ + 3.463[+(x-3082)]_+ + 1.6032[+(x-1209)]_+ - 1.6226[+(x-878)]_+ - 1.3511[+(x-2868)]_+ - 0.5161[+(x-1412)]_+$

FIGURE 21. New Fit of Pozzetti et al's "Signature"

- Required Number of Points: 102
- Estimated Accuracy (EA): 7.6%
- MARS Model: $y = 0.0551 + 3.2048[-(x-2982)]_+ + 0.7094[+(x-1985)]_+ - 23.1841[-(x-1281)]_+ + 23.2056[+(x-1281)]_+ - 19.9379[+(x-989)]_+ + 0.6006[-(x-3958)]_+ + 19.425[-(x-1006)]_+ + 21.3061[+(x-3013)]_+ + 24.5915[+(x-2995)]_+ - 0.5823[-(x-2868)]_+$

FIGURE 22. Residuals as a Percentage of the Observed Values

FIGURE 23. Percentage Difference between 1000 True Points and the Model:

Pozzetti's signature is the most complex of the four functions and so required the most points to fit. Nevertheless, the new implementation of MARS is able to approximate it very well. The estimated accuracy is greater than the 99th percentile of percentage deviations between the MARS model and the true function at the 1000 points which were sampled. The disadvantage to eliminating the required minimum span of observations between knots is that segments of the model become vulnerable to bias should there be a consecutive run of observations either below or above their means. In low noise experiments, however, this should not present a serious problem.

5.3 Summary

The *EA* validation tests demonstrated that the accuracy estimate, based on the pseudo-confidence interval, is realistic although somewhat conservative. The construction process validation tests demonstrated that the incremental technique is capable of building resource functions to within a required accuracy for a variety of underlying, known functions. Furthermore, we noticed that the point selection algorithm distributes the design points in a way that favours regions of high percentage deviation. Friedman's implementation of MARS had trouble fitting discontinuous functions although the new implementation worked quite well. The weakness of the new approach is that it is

5.3 Summary

vulnerable to runs in which the measurements are either below or above their mean.

There is a trade-off which exists, therefore, between a model being insensitive to measurement error and sensitive to discontinuities in the underlying function. A possible enhancement to the smartexec script would be to let the user choose which modeling technique to use. The following chapter presents a case study to test how well the incremental construction process works on real software components.

Case Study: The ACE Event Server

6.0 Overview

The smartexec feature of the enhanced Performance Workbench was used to develop resource functions for the components of the ACE Event Server. This chapter first describes the ACE Event Server and its corresponding layered queuing model. It then explains the measurement harnesses which were employed in the experiments. Finally it presents the MARS models which were developed using the incremental technique and examines the results of the performance models whose execution times are based on these resource functions. Friedman's implementation of MARS was used to construct the resource functions.

6.1 The ACE Event Server

6.1.1 The Components Involved

The ACE Event Server allows a group of supplier processes to pass messages to a group of consumer processes. It can be used, for instance, by a group of clients which want to be notified of certain events. To facilitate group communications, a number of ACE classes[Schmidt93] are employed: a Reactor, a Socket Acceptor, a Service handler and a user-level ASX Stream.

- **The ACE Reactor**

The ACE Reactor[Schmidt94a] is just an object-oriented wrapper for the select system call. It allows a Service Handler object to register an interest in a communication endpoint - represented by an I/O handle. When the Reactor's `handle_event` method is invoked, it issues a select call on the registered I/O handles. When input arrives on one of those handles, the Reactor "calls back" the Service Handler object which was interested in that handle.

FIGURE 24. The Reactor

- **The ACE Svc_Handler**

Application-specific event handling is implemented within a Service Handler class. All Service Handler classes inherit from the ACE Svc_Handler base class and must implement the `handle_input` and an `open` method. The Reactor invokes `handle_input` as part of the call-back. Each Service Handler owns a peer object which is essentially an object wrapper for its communication endpoint.

FIGURE 25. An ACE Service Handler

- **The ACE Acceptor**

An ACE Acceptor[Schmidt96] is a special subclass of `Svc_Handler` whose peer object is a passive socket. An Acceptor object is initialized with the class of Service Handler to instantiate when a connection request arrives on the passive socket. An Acceptor object registers itself and the descriptor of the passive socket with the Reactor. When a connection request arrives on that socket, the Reactor calls back the Acceptor object. The Acceptor object then instantiates an object of the Service Handler class with which it was initialized. It then invokes the `open` method of the new Service Handler object. It is up to the Service Handler object to register itself and the descriptor of its active socket with the Reactor.

- **The ACE SOCK_Stream and SOCK_Acceptor**

An ACE SOCK_Stream object[Schmidt92] provides an object-oriented interface to a Unix active stream socket while a SOCK_Acceptor provides an OO interface to a passive socket. Most Service Handler objects have a SOCK_Stream object as their peer. An Acceptor object has a SOCK_Acceptor object as its peer.

- **The ACE ASX Stream, Module, Tasks, and Message_Blocks**

An ASX Stream class[Schmidt94b] provides a flexible architecture for assembling a number of functions together through which data will pass sequentially. An ACE Stream object contains an application-specified number of ACE Module objects which are pushed onto it at run time. Each ACE Module object is comprised of two ACE Task objects. One of the Task objects accepts, processes, and passes messages downstream, while the other Task does the same thing upstream. ACE Task objects can either be active, in which case they have their own message queue[Schmidt95], or passive. Messages are passed along the stream using ACE Message_Block objects. A Message_Block object simply stores a pointer to opaque data as well as maintains a read and write pointer to that data.

FIGURE 26. ASX Stream, Module, Task.

6.1.2 Putting the Components Together in the ACE Event Server

When the event server is executed, it first creates an ASX Stream object and pushes three Modules onto it. The first Module contains an ACE Supplier_Router Task object; the middle one contains an ACE Event_Analyzer Task object; and the last module contains a Consumer_Router Task object. All the Tasks are subclasses of the ACE Task. The Consumer_Router Task object is active. When a supplier process connects to the event server, the Acceptor object instantiates a supplier Service Handler for that connection. The supplier Service Handler then records a pointer to the Supplier_Router Task and registers itself with the Reactor. When a consumer process connects to the event

server, a consumer Service Handler is instantiated and stored in a list containing all the consumer Service Handlers. When a message arrives from a supplier, the Reactor calls back that supplier's Service Handler. The Service Handler reads in the message from its SOCK_Stream object, formats it into a Message_Block object and passes it to the Supplier_Router Task. The Message_Block object is moved upstream until it is placed in the Consumer_Router Task's message queue. When the Consumer_Router Task is ready, it reads the Message_Block off its queue, extracts the message and iterates through the list of consumer Service Handlers sending the message to each of their SOCK_Stream peers. Figure 27, "Message Broadcast Through ACE Event Server," on page 119 graphically depicts a message flowing from a supplier to a group of consumers through the ACE Event Server.

FIGURE 27. Message Broadcast Through ACE Event Server

6.1.3 Call Structure and Threading of the ACE Event Server

From a threading point of view, the ACE Event Server is composed of two active sections. The main thread runs through the reactor and is responsible for reading in an message from the supplier. We refer to this thread as the supplier handler thread. Once

the message has been read in, the receiving thread runs through the stream up to and including the Event Analyzer module. It then drops the message off on to the message queue for the Consumer Router active object.

FIGURE 28. Call Structure and Active Objects

Figure 28, “Call Structure and Active Objects,” on page 120 shows the general sequence of calls which take place after the Reactor has called back a supplier’s Service Handler

when a message arrives from that supplier. The `handle_input` method is invoked from the Reactor. The Service Handler in turn invokes `SOCK_Stream`'s `recv_n` method to read in the message and then calls the `reply` method of the `Supplier_Router`'s Task. This method ends up calling the `put` method of the `Event_Analyzer` Task which in turn calls the `put` method of the `Consumer_Router` Task. The `put` method of the `Event_Analyzer` executes the main body of that Task whereas the `put` method of `Consumer_Router` simply places the message on the `Consumer_Router` Task's message queue.

Within the `Consumer_Router` thread, the `svc` method cycles continuously, retrieving messages off the queue and invoking the `send_peers` method.

`Peer_Router::send_peers` iterates through the consumer Service Handlers calling `Peer_Handler::put` for each one. The number of times `Peer_Handler::put` is called by `Peer_Router::send_peers`, therefore, depends on the number of consumer processes registered. The `Peer_Handler::put` method simply calls `SOCK_Stream`'s `send_n` method for that Service Handler's connection. The result is that the message is sent to all the registered consumer processes.

6.1.4 A Performance Model for the ACE Event Server

The active heterogeneous threads within the Event Server naturally lend themselves to being modeled as separate tasks - one task for the thread executing the reactor and supplier service handler code and the other to cycle within the Consumer_Router Task's svc method. A layered queueing model of this application, with two supplier processes, therefore, will look like that in Figure 29, "Layered Queuing Model of ACE Event Server," on page 123. The supplier processes act as clients to the system which generate requests. A network task has been added to model network contention and propagation delay. Consumer processes have not been included since it is assumed that the only work they do is to receive messages. As such, the consumers would not bottleneck the system. We will now construct the resource functions for the supply handler and consumer router in order to provide execution time estimates in the layered model.

FIGURE 29. Layered Queuing Model of ACE Event Server

6.2 The Experiments and Resource Functions

There is of course, no true function against which to compare the MARS models which are shown in this section. But to get an idea of the accuracy, after the experiments were run, new points were randomly selected from the design space and 30 replicated measurements were made at each point. The averages of these replications were then compared to the MARS model estimates. For the supplier handler resource function, which only involved 1 independent parameter, 10 new points were selected. For the consumer router resource function, which was based on 2 independent parameters, 20 points were randomly chosen.

6.2.1 Supplier Handler

Since the supplier handler only reads in messages and adds them to the consumer router's queue, its CPU demand is only dependent on the size, in bytes, of the message received. To measure the cpu time required by the supplier handler to receive and process one message, a modified version of the Event Handler was constructed in which the Consumer Router was replaced by a Dummy Task. This task provided no service for its queue and so the only work done in the application was by the supplier handler thread. A harness was constructed to launch an event handler along with a supplier which would send it a certain number of messages. The number of messages, the size of the message in bytes, and the machine locations of both the supplier and event handler were user-specified. In these experiments, the user was the "smartexec" script.

The range of possible message sizes was between 1 and 64000 bytes with a granularity of 1. 64K was chosen as an upper limit because that is the largest socket buffer permitted by the internet protocol. During each experiment run, a given message was sent 200 times and the CPU time per message was taken as the average. Since each measurement involves some error, a smoother of 3 was used and the accuracy requirement was set to 10 %. The server on which measurements were performed was launched on one UltraSparc(sunspot) and the supplier was run on another UltraSparc(suntan). The

observed values and the resulting MARS resource function are shown in Figure 30, “MARS Resource Function for the Supplier Handler,” on page 126.

FIGURE 30. MARS Resource Function for the Supplier Handler

- Required Number of Points: 26
- Estimated Accuracy (EA): 9.68%
- MARS Model: $CPU\ Demand = 871.475 + 0.0312[+(msgsize-8772)]_+ - 0.0473[-(msgsize-8772)]_+$

FIGURE 31. Residuals as a Percentage of the Observed Values

TABLE 2. Means from 30 Replications vs MARS Model Estimate, for 10 Cases

Message Size(Bytes)	Mean from 30 Replications	MARS Model Estimate	%age Difference
1	436.833	457.011	4.62
5830	756.567	732.454	-3.18
15267	1127.667	1073.81	-4.78
27207	1474.033	1445.76	-1.92
27871	1336.967	1466.45	9.68
48228	2015.400	2100.6	4.23
50759	2308.567	2179.45	-5.59
50797	2245.500	2180.63	-2.89
58031	2398.833	2405.99	-0.30
64000	2593.400	2591.93	-0.06

The resource function estimates for each randomly selected design point all fall within 10% of the means from the replications.

6.2.2 Consumer Router

The CPU demand of the consumer router is dependent on both the size of the message to broadcast and the number of consumers registered in the broadcast group. As such an experiment was set up to allow one supplier to send a given sized message, a given number of times, to a given number of consumers. As in the last experiment each message was sent 200 times. The size of the message and the number of consumers in each run was determined by the “smartexec” script. To measure the CPU consumption of just the Consumer Router, the Event Server was again modified, this time to allow the supplier to send all the required number of messages at once. After the supplier handler had loaded up the consumer router’s queue, the instrumentation was then turned on to measure how much cpu was required by the consumer router to process all the messages. The cpu time per message was calculated as the average.

By the same reasoning used for the supplier handler experiment, the message size was varied between 1 and 64000 bytes with a granularity of 1 byte. The domain for the number of consumers was specified between 1 and 10, with a granularity of 1.(You cannot have .5 consumers). 10 was chosen as an upper limit because we assume this broadcast server, built on top of a unicast protocol, can only be used by small groups of

processes. If the application was going to be used by larger groups, then, of course, the upper limit would be set by the needs of the application. The required accuracy was, again, set at 10%. The event handler was launched on an UltraSparc(sunspot) while the supplier and consumers were distributed over four UltraSparcs(suntan, sunrise, sunspot, and sunset). The resulting MARS resource function along with the observed values are shown in Figure 32, “MARS Resource Function for the Consumer Router,” on page 130.

FIGURE 32. MARS Resource Function for the Consumer Router

- Required Number of Points: 157
- Estimated Accuracy (*EA*): 8.58%
- MARS Model: $CPU = 1436.73 + 0.1314[+(msgsize-1)]_+ - 0.0159[-(consumers-9)]_+ [- (msgsize-1)]_+ + 808.082[+(consumers-9)]_+ - 149.399[-(consumers-9)]_+ - 0.03[+(consumers-9)]_+ [- (msgsize-21091)]_+ + 0.0092[+(consumers-1)]_+ [- (msgsize-10808)]_+ + 0.0989[+(msgsize-4223)]_+ + 0.01[-(consumers-9)]_+ [+(msgsize-5424)]_+$

FIGURE 33. Residuals as a Percentage of the Observed Values

TABLE 3. Means from 30 Replications vs MARS Model Estimate, for 20 Cases

Message Size(Bytes)	# Consumer Processes	Mean from 30 Replications	MARS Model Estimate	%age Difference
1	1	223.833	241.538	7.91
1	10	686.867	713.906	3.94
510	5	477.067	493.452	3.43
2742	4	580.200	609.302	5.02
4342	2	445.233	431.805	-3.02
6111	4	973.867	1030.88	5.85
11754	3	1224.067	1333.38	8.93
20461	3	1959.700	1989.95	1.54
26662	6	4797.233	4809.16	0.02
27497	3	2527.200	2520.51	-0.03
30663	2	1918.267	1872.05	-2.41
36253	4	4062.133	4212.35	3.70
40655	8	8640.100	9239.01	6.93
47916	2	2718.767	2727.47	0.03
50530	8	11035.900	11258.8	2.10
52249	6	8629.933	8721.05	1.06
52412	6	8605.233	8745.97	1.64
57666	5	7574.767	7964.64	5.15
64000	1	1664.833	1776.75	6.72
64000	10	16800.767	16570.2	-1.37

All the MARS estimates are within 10% of the means from the replications. 95% are within the estimated accuracy of 8.58%. From the experiments run on both the supplier handler and consumer router then, we can conclude that the incremental technique does achieve a close fit to the true function and the GCV-based accuracy estimate does provide a reasonably good idea of the model's deviation from the true function.

6.2.3 Comparison with Simple Linear Regression

Fitting the data using simple linear regression will provide a clear justification for why the technique proposed in this thesis is necessary. In the case of the supplier handler, a linear model is adequate for modeling its resource function. It's formula is simply: $CPU = 543.281 + 0.0325msgsize$. In the case of the consumer router, however, a simple linear model is not a viable option. The following figures display the linear fit on the Consumer Router measurements along with a histogram of percentage deviations.

FIGURE 34. Linear Resource Function for the Consumer Router

- Linear Model: $CPU = -2550.76 + 0.12336msgsize + 591.483consumers$

FIGURE 35. Linear's Residuals as a Percentage of the Observed values

Over 5% of residuals have a greater than 450% percent deviation from the linear model. Not only that, but many of the cpu time estimates, for small numbers of consumers and small message sizes, are actually negative. Clearly, then, the linear modeling technique is not sufficient for constructing even a simple resource function with 2 independent parameter. By using a flexible modeling technique combined with an accuracy estimate based on percentage deviation and a point selection algorithm which concentrates points in areas of high deviation the technique proposed in this thesis is able to provide a close fit and avoid unreasonable percentage deviations like those witnessed above.

6.3 Impact of Execution Time Estimates on the Performance Model

In this section we examine how the inputs from the linear and MARS-based resource functions affect the throughput of the performance model introduced earlier. Suppose we wish to model the broadcast of a 200 byte message to 10 consumer processes. (These numbers have been chosen to highlight the difference between the two resource functions). The suppliers are scattered across 4 machines (when there are enough suppliers) and each supplier has been given an arbitrary, exponentially distributed think time of 4 millisecs. The network “task” is a delay server with a delay of .16 milliseconds. The supplier handler task is given a multiplicity of 9 to model the fact that the Consumer Router has a queue which can only contain up to 8 waiting messages. Using this technique, both the supply handler and consumer router execute in the first phase and their execution times were made deterministic. The estimated execution times from both the linear and MARS resource functions are:

TABLE 4. Estimated Service Time, 200 Byte Message, 10 Consumers

Type of Resource Function Model	Supplier Handler Execution Time (us)	Consumer Router Execution Time (us)
Linear	549.781	3388.742
MARS	466.415	762.568

Figure 36, “Comparing the Two Throughputs,” on page 136 demonstrates how the two throughputs change as the supplier count increases from 1 to 10.

FIGURE 36. Comparing the Two Throughputs

Assuming the throughputs for the MARS model are correct, if you are not willing to tolerate up to 40% error in your throughput estimate, then the simple linear resource function will not satisfy your accuracy requirements. The interesting thing here is that, while the linear resource function’s estimate of execution time for the Consumer Router is 344% more than the estimate from the MARS resource function, the throughput of the model based on the linear inputs is at most 38% less than that of the model based on the MARS inputs. In other words, the elasticity of the model throughput with respect to

execution time is much less than 1. If we know the elasticity of the model with respect to execution time, then, we can work backwards to determine the required accuracy of our resource functions. As mentioned earlier, either manual perturbations or Sheikh's technique[Sheikh97] can be used to determine the elasticity. In this case, if we ignore the effect of the change in the Supplier Handler execution time on throughput, the elasticity of the model's throughput with respect to Consumer Router execution time is roughly -0.11 i.e. (-38%/344%). If we required no more than 5% error in the throughput due to error in the execution time estimate, then we could tolerate our resource function differing by up to 45% from the true function. Now suppose we had also considered the elasticity of the throughput with respect to the Supplier Handler execution time, and it turned out to be -0.11 as well. In this case, an error of no more than 5% in the throughput could be guaranteed by setting the required accuracy of both resource functions to 22.5% or less. In either case, the resource functions are already within 10% accuracy.

7.0 Success

The technique described in this thesis has been able to incrementally build a resource function in a way that progresses towards the accuracy required by the user. It has been successfully deployed on a series of analytical functions with noise and has also been used to develop resource functions for the tasks in the ACE Event Server. The validation tests and the case study demonstrated the following:

- The accuracy estimate, *EA*, works. It provides a realistic, though somewhat conservative, estimate of the percentage deviation between the MARS model and the true function. The probability that the MARS model deviates by no more than *EA* from the true model is at least 90%.

- Using the accuracy estimate as a guide, the number of points collected is controlled well. A large number of points is only required if the underlying function is complex or the accuracy requirement is stringent.
- The MARS models are able to fit non-linear functions as well as jagged functions provided the requirement of a minimum span of observations between knots is eliminated.

This technique, therefore, simplifies the process of constructing a resource function. The modeler must, of course, instrument the code and provide the range of values for each independent parameter, but the experiment itself is “designless”. The design is determined dynamically by the “smartexec” script as more measurement points are taken.

The incremental process is able to construct resource functions to within a specified accuracy, using only as many measurements as are required to achieve that accuracy. In this thesis we have suggested using the accuracy requirements of the performance model in which the resource function will be employed to establish the required accuracy of the resource function. Given the elasticity of the performance model responses with respect to their parameters, we can work backwards to determine how accurate the parameter estimates need to be. By only taking as many points as are needed to satisfy that accuracy, the incremental construction process is able to build resource

functions with “good enough” fidelity for the purpose of performance modeling. If the performance model context is unknown at the time the resource function is built, a reasonable, arbitrary accuracy requirement must be set. In this thesis we have used an accuracy requirement of 10%. When that resource function is used within a performance model, its accuracy along with the elasticity of the performance model responses can be used to gauge what percentage error can be expected in the performance model predictions.

7.1 Caveats and Future Work

The technique has not been tested with more than 2 independent parameters nor has it been used to fit measurements with very high random measurement errors. Different point selection algorithms could also be examined. The current algorithm is quite sensitive to high percentage deviations. In experiments with low random measurement error, where bias is the largest contributor to error, such a strategy is appropriate. It may not, however, be appropriate with high measurement errors. All these areas offer an opportunity for future research.

The log transformations used in calculating the pseudo-confidence intervals only work if the resource demands are greater than 0. Since most measurements of software resource demand are greater than 0, this is not usually a problem. However, if observations could take on zero values, a new scheme would be needed to estimate the accuracy. The important thing is to base the scheme on residuals as a percentage difference and not actual distance between the observations and the model. In this way, large percentage differences result in wider pseudo-confidence intervals.

-
-
- [Booth80] Booth, T.L. and Wiecek, C.A., "Performance Abstract Data Types as a Tool in Software Performance Analysis and Design", *IEEE Transactions on Software Engineering*, Vol.6, No.2, 1980, pp.138-151.
- [Box87] Box, G.E.P., *Empirical Model Building and Response Surfaces*, John Wiley, 1987.
- [Efron79] Efron, B., "Bootstrap Methods: Another Look at the Jackknife", *Annals of Statistics*, Vol. 7, No. 1, 1979, pp.1-26.
- [Efron81] Efron, B., "Nonparametric Standard Errors and Confidence Intervals", *The Canadian Journal of Statistics*, Vol. 9, No. 2, 1981, pp.139-158.
- [Ellis90] Ellis, M.A. and Stroustrup, B., *The Annotated C++ Reference Manual*, Addison-Wesley, 1990
- [Friedman89] Friedman, J.H., "Flexible Parsimonious Smoothing and Additive Modeling", *Technometrics*, Vol. 31, No. 1, 1989, pp.3-39.
- [Friedman91] Friedman, J.H., "Multivariate Adaptive Regression Splines", *Annals of Statistics*, Vol. 19, No. 1, 1991, pp.1-141.
- [Gordon80] Gordon, K.D. and Dowdy, L.W., "The Impact of Certain Parameter Estimation Errors in Queuing Network Models", in: *Proceedings of the Performance 80 Conference; Performance Evaluation Review*, Vol.9, No. 2, 1980, pp.3-9.
- [Hastie90] Hastie, T.J. and Tibshirani, R.J., *Generalized Additive Models*, Chapman, 1990
- [Khuri87] Khuri, A.I. and Cornell, J.A., *Response Surfaces, Design and Analyses*, Marcel Dekker, 1987
- [Libes95] Libes, D., *Exploring Expect - a Tcl-based Toolkit for Automating Interactive Programs*, O'Reilly & Associates, Inc., 1995

-
-
- [Menasce97] Menasce, D., "A Framework for Software Performance Engineering of Client/Server Systems", to appear in CMG'97, Orlando, Florida
- [Myers89] Myers, R.H., et al., "Response Surface Methodology: 1966-1988", *Technometrics*, Vol. 31, No. 2, 1989, pp.137-157.
- [Neter74] Neter, J. and Wasserman W., *Applied Linear Statistical Models, Regression, Analysis of Variance, and Experimental Designs*, Richard D. Irwin, 1974.
- [Ousterhout94] Ousterhout, J.K., *Tcl and the Tk Toolkit*, Addison-Wesley, 1994
- [Pozzetti95] Pozzetti, E. et al., "Characterizing the Resource Demands of TCP/IP", in: *Proceedings of the 1995 Conference on High Performance Computing and Networking (HPCN95)*, May, 1995.
- [Prenter75] Prenter, P.M., *Splines and Variational Methods*, John Wiley, 1975.
- [Press88] Press, W.H., et al., *Numerical Recipes in C, The Art of Scientific Computing*, Cambridge University Press, 1988
- [Rolia95] Rolia, J.A. and Sevcik, K.C., "The Method of Layers", *IEEE Transactions on Software Engineering*, Vol. 21, No.8, 1995, pp. 689-700.
- [Schmidt92] Schmidt, D.C., "IPC_SAP: An Object-Oriented Interface to Interprocess Communication Services", *C++ Report*, Nov/Dec 1992
- [Schmidt93] Schmidt, D.C., "The ADAPTIVE Communication Environment: Object-Oriented Network Programming Components for Developing Client/Server Applications", *11th Sun User Group Conference*, San Jose, Dec, 1993
- [Schmidt94a] Schmidt, D.C., "Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching", *Proceedings of the First Pattern Languages of Programs Conference*, August 1994

-
-
- [Schmidt94b] Schmidt, D.C., "ASX: An Object-Oriented Framework for Developing Distributed Applications", *6th USENIX C++ Conference*, April 1994
- [Schmidt95] Schmidt, D.C., "An OO Encapsulation of Lightweight OS Concurrency Mechanisms in the ACE Toolkit", Washington University technical report WUCS-95-31
- [Schmidt96] Schmidt, D.C., "Acceptor and Connector: A Family of Object Creational Patterns for Initializing Communication Services", First European Conference on Pattern Languages of Programs, Kloster Irsee, July 1996
- [Sheikh97] Sheikh, F., Master's Thesis in progress, University of Carleton
- [Vetland93] Vetland, V., *Measurement-Based Composite Computational Work Modelling of Software*, PH.D. Thesis, University of Trondheim, April 1993
- [Vetland96] Vetland, V. and Woodside, M., "A Workbench for Automation of Systematic Measurement of Resource Demands of Software Components", Carleton University technical report SCE-96-06.
- [Wahba90] Wahba, G., *Spline Models for Observational Data*, Society for Industrial and Applied Mathematics, 1990.
- [Woodside95] Woodside, C.M., et al., "The Stochastic Rendezvous Network for Performance of Synchronous Client-Server-Like Distributed Software", *IEEE Transactions on Computers*, Vol. 44, No.1, 1995, pp.20-34
- [Xu96] Xu, Z. and Hwang, K., "Modeling Communication Overhead: MPI and MPL Performance on the IBM SP2", *IEEE Parallel and Distributed Technology*, Spring 1996, pp.9-23.

IMAGE EVALUATION TEST TARGET (QA-3)

APPLIED IMAGE, Inc.
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved