

**BUILDING A KNOWLEDGE BASED SIMULATION OPTIMIZATION
SYSTEM WITH DISCOVERY LEARNING**

by

Fernando C. Siochi

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

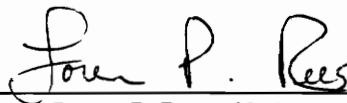
in partial fulfillment of the requirements for the degree of

Doctorate of Philosophy

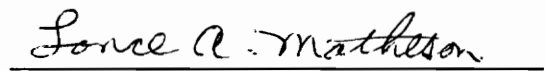
in

Management Science

APPROVED



Loren P. Rees, Chairman


Edward R. Clayton
Lance A. Matheson
Cliff T. Ragsdale
Terry R. Rakes

**November 1, 1995
Blacksburg, Virginia**

Key words: Simulation Optimization, Expert Systems, Discovery Learning, Best-First Search

C.2

LD
5655
V856
1995
S563
C.2

BUILDING A KNOWLEDGE BASED SIMULATION OPTIMIZATION SYSTEM WITH DISCOVERY LEARNING

by

Fernando C. Siochi

Loren P. Rees, Chairman

Management Science

(ABSTRACT)

Simulation optimization is a developing research area whereby a set of input conditions is sought that produce a desirable output (or outputs) to a simulation model. Although many approaches to simulation optimization have been developed, the research area is by no means mature.

This research makes three contributions in the area of simulation optimization. The first is fundamental in that it examines simulation outputs, called "response surfaces," and notes their behavior. In particular both point and region estimates are studied for different response surfaces: Conclusions are developed that indicate when and where simulation-optimization techniques such as Response Surface Methodology should be applied.

The second contribution provides assistance in selecting a region to begin a simulation-optimization search. The new method is based upon the artificial intelligence based approach best-first search. Two examples of the method are given.

The final contribution of this research expands upon the ideas by Crouch for building a "Learner" to improve heuristics in simulation over time. The particular case of parameter-modification learning is developed and illustrated by example.

The dissertation concludes with limitations and suggestions for future work.

ACKNOWLEDGMENTS

I give the praise and the glory to my Lord and God Jesus for sustaining me in this effort. I thank Him for providing me with wisdom and counsel in all matters.

I thank my advisor, Dr. Loren Paul Rees, for his time and encouragement. I have learned many lessons about academia and life from him. He provided invaluable insight and guidance in the process of writing and developing ideas for the dissertation.

I appreciate the time and effort that Dr. Clayton, Dr. Matheson, Dr. Ragsdale and Dr. Rakes put in while serving on my committee and for sharing their knowledge and experience.

I wish to express my gratitude to Dr. Taylor for the support of my studies and for the opportunity to teach at the university level.

I would like to acknowledge the support and camaraderie of my fellow students, Rene, Ingrid, Barry, Mike, Jay, Jack, and Mark for making student life a little easier to bear. I also appreciate the friendship of Rene, Alan, and John Paul, who helped me keep in touch with the things that really mattered.

I am grateful for my parents Andres and Loiva for their support and encouragement. They have taught me much about excellence and the pursuit thereof. They instilled in me not only the desire to learn and to achieve, but to do so and give the glory and honor to my Lord Jesus Christ.

I appreciate the love, support and patience of my wife Tina, for enduring with me through the process. I am grateful for her understanding the demands on my time and I treasure her words of encouragement. Tina has a way of putting things in perspective and enabling me to continue with the work at hand.

TABLE OF CONTENTS

CHAPTER ONE: INTRODUCTION	1
SIMULATION	1
SIMULATION AND SIMULATION OPTIMIZATION ISSUES	4
KNOWLEDGE-BASED SYSTEMS	6
MACHINE LEARNING	9
KNOWLEDGE DISCOVERY AND DISCOVERY SYSTEMS	10
KNOWLEDGE-BASED SIMULATION OPTIMIZATION	12
PURPOSE OF RESEARCH	14
SCOPE AND LIMITATIONS	18
PLAN OF PRESENTATION	19
CHAPTER TWO: LITERATURE REVIEW	20
SIMULATION	20
SIMULATION OPTIMIZATION	23

TYPICAL ASSUMPTIONS	25
KNOWLEDGE-BASED SIMULATION OPTIMIZATION	26
Learning: Definitions, Advantages, and What There is to Learn	30
The Crouch Learner.....	35
KNOWLEDGE DISCOVERY.....	40
KNOWLEDGE DISCOVERY IN THE SIMULATION OPTIMIZATION DOMAIN	42
Data Dictionary	44
Sessions History Database.....	44
Domain-Knowledge Module	47
Discovery Methods module.....	53
CHAPTER THREE: AN INVESTIGATION OF THE BEHAVIOR OF SIMULATION RESPONSE	
SURFACES.....	54
INTRODUCTION.....	54
DEFINITION OF THE SIMULATION OPTIMIZATION PROBLEM	56
DEFINITION OF THE EXEMPLARY MODEL AND EXPERIMENTS.....	58
Simulated inventory system	58
Experimental conditions	63
Data Collection Scheme.....	65
STATISTICAL MEASURES OF THE BEHAVIOR OF SIMULATION SURFACES	67
Point-estimate Measures	67
Location Measures	68
Dispersion Measures	68
Relative Activity	69
Regional Measures.....	70

Test for Normality	71
Test for Homogeneous Variances.....	72
F-test for Significance of Regression.....	77
F-test for Lack of Fit	77
EXPLORATION	78
Point Estimates.....	79
Mean	79
Standard Deviation.....	79
Coefficient of Variation	84
Signal to Noise Ratio.....	84
Relative Activity or “Bumpiness.”	89
Regional Estimates	92
Test for Normality.....	92
Homogeneity of Variance	99
“Searchable” Regions	99
CONCLUSIONS	106
CHAPTER FOUR: A BEST-FIRST SEARCH APPROACH FOR DETERMINING STARTING	
REGIONS IN SIMULATION OPTIMIZATION	109
INTRODUCTION.....	109
Definition of Simulation Optimization.....	109
Typical Assumptions	112
ILLUSTRATIVE EXAMPLE.....	114
BEST-FIRST SEARCH.....	118
STARTING A SIMULATION OPTIMIZATION SEARCH.....	122
Objectives	122

Implementation of the Starter	128
EXAMPLES	132
Example 1: Inventory Model with Low-Variance Demand and No Lead Time	133
Example 2: Inventory Example with Higher Demand Variance and Lead Time Variance	150
Example 3: A Multimodal Response Surface	160
CONCLUSIONS AND FUTURE RESEARCH.....	162
CHAPTER FIVE: BUILDING A KNOWLEDGE-BASED SIMULATION OPTIMIZATION	
SYSTEM WITH DISCOVERY LEARNING.....	165
BACKGROUND	165
Knowledge-based Simulation Optimization	165
Learning: Definitions, Advantages, and What There is to Learn	168
The Crouch Learner.....	174
Overview	174
Crouch Process Flow	176
KNOWLEDGE DISCOVERY	178
KNOWLEDGE DISCOVERY IN THE SIMULATION OPTIMIZATION DOMAIN	181
Data Dictionary	183
Sessions History Database.....	184
Domain-Knowledge Module	185
Discovery Methods module	192
BUILDING A SYSTEM: A PARAMETER MODIFICATION EXAMPLE	193
CONCLUSIONS AND FUTURE WORK.....	201
CHAPTER SIX: CONCLUSIONS.....	203

SURFACE STUDY	203
BEST FIRST SEARCH APPROACH	206
BUILDING A KBSOS WITH DISCOVERY LEARNING	208
BIBLIOGRAPHY	210
APPENDIX A	214
APPENDIX B	219
VITA	225

List of Figures

FIGURE 1.1. EXPERT SYSTEM ARCHITECTURE	8
FIGURE 1.2. THE FRAWLEY ET AL DISCOVERY PARADIGM	10
FIGURE 1.3. THE SIMULATION-OPTIMIZATION PROCESS	12
FIGURE 1.4. GREENWOOD-REES-CROUCH SIMULATION-OPTIMIZATION ARCHITECTURE.....	15
FIGURE 1.5. VISUALIZATION OF THE LEARNER AND ITS ENVIRONS	16
FIGURE 2.1. THE SIMULATION-OPTIMIZATION PROCESS.....	27
FIGURE 2.2. GREENWOOD-REES-CROUCH SIMULATION-OPTIMIZATION ARCHITECTURE.....	29
FIGURE 2.3. VISUALIZATION OF THE LEARNER AND ITS ENVIRONS	31
FIGURE 2.4. AN OVERVIEW OF THE CLASSIFIER KBSOS	33
FIGURE 2.5. CROUCH'S LEARNING PROCESS (CROUCH, 1992).....	38
FIGURE 2.6. THE FRAWLEY ET AL DISCOVERY PARADIGM	41
FIGURE 2.7. THE DISCOVERY LEARNER AND ITS INTERACTION WITH THE CLASSIFIER KBSOS.....	43
FIGURE 2.8. A LATTICE SHOWING THE INTERCONNECTIONS OF THE SESSIONS HISTORY DATABASE FRAMES ...	45
FIGURE 2.9A. CONCEPTS FRAME	46

FIGURE 2.9B. SESSIONS FRAME	46
FIGURE 2.9C. RULE FRAME	46
FIGURE 2.9. EXAMPLES OF FRAMES	46
FIGURE 2.10. DETAILS OF THE DOMAIN KNOWLEDGE COMPONENT OF THE DISCOVERY LEARNER	50
FIGURE 2.11A. INITIAL LINK-OF-INFLUENCE.....	52
FIGURE 2.11B. SUBSEQUENT LINK-OF-INFLUENCE.....	52
FIGURE 2.11. LINK-OF-INFLUENCE	52
FIGURE 3.1. SIMPLE INVENTORY MODEL THAT PERMITS BACK ORDERS AND EXHIBITS BOTH STOCHASTIC DEMAND AND LEAD TIME.....	59
FIGURE 3.2. PROCESS FOR OPTIMIZING THE SIMULATED INVENTORY SYSTEM.....	62
FIGURE 3.3A. Δ -BY- Δ REGIONS.....	65
FIGURE 3.3B. 2Δ -BY- 2Δ REGIONS	65
FIGURE 3.3. INVENTORY MODEL'S DECISION SPACE	65
FIGURE 3.4. THREE-DIMENSIONAL PLOTS OF THE MEAN OF THE RESPONSE	80
FIGURE 3.5. CONTOUR PLOTS OF THE MEAN OF THE RESPONSE FOR CASES LNB AND HMW	81
FIGURE 3.6. THREE-DIMENSIONAL PLOTS OF THE STANDARD DEVIATION OF THE RESPONSE	82
FIGURE 3.7. HISTOGRAMS OF THE STANDARD DEVIATION OF THE RESPONSE.....	83
FIGURE 3.8. THREE-DIMENSIONAL PLOTS OF THE COEFFICIENT OF VARIATION OF THE RESPONSE.....	85
FIGURE 3.9. HISTOGRAMS OF THE COEFFICIENT OF VARIATION OF THE RESPONSE.....	86
FIGURE 3.10. THREE-DIMENSIONAL PLOTS OF THE SIGNAL-TO-NOISE RATIO OF THE RESPONSE	87
FIGURE 3.11. HISTOGRAMS OF THE SIGNAL-TO-NOISE RATIO OF THE RESPONSE	88
FIGURE 3.12. SECOND DIFFERENCES OF THE RESPONSE WITH Q HELD CONSTANT.....	90
FIGURE 3.13. SECOND DIFFERENCES OF THE RESPONSE WITH R HELD CONSTANT	91
FIGURE 3.14. SECOND DIFFERENCES OF THE RESPONSE WHEN THE FIRST DIFFERENCE CROSSES ZERO; WITH Q HELD CONSTANT	93

FIGURE 3.15. SECOND DIFFERENCES OF THE RESPONSE WHEN THE FIRST DIFFERENCE CROSSES ZERO; WITH R HELD CONSTANT.....	94
FIGURE 3.16. THREE-DIMENSIONAL PLOTS AND HISTOGRAMS OF P-VALUES FOR NORMALITY TEST, $\Delta=20$	95
FIGURE 3.17. THREE-DIMENSIONAL PLOTS AND HISTOGRAMS OF P-VALUES FOR NORMALITY TEST, $\Delta=40$	96
FIGURE 3.18A. THE DISTRIBUTION OF RESIDUALS FOR A "TYPICAL" \mathcal{R}_6 ($\Delta=40$).....	98
FIGURE 3.18B. THE DISTRIBUTION OF RESIDUALS FOR A "TYPICAL" \mathcal{R}_3 ($\Delta=20$).....	98
FIGURE 3.18. THE DISTRIBUTION OF RESIDUALS FOR DIFFERENT INTER-GRIDPOINT SPACINGS.	98
FIGURE 3.19. THREE-DIMENSIONAL PLOTS AND HISTOGRAMS OF P-VALUES FOR HOMOGENEITY-OF- VARIANCE TEST, $\Delta=20$	100
FIGURE 3.20. THREE-DIMENSIONAL PLOTS AND HISTOGRAMS OF P-VALUES FOR HOMOGENEITY-OF- VARIANCE TEST, $\Delta=40$	101
FIGURE 3.21. LOCATIONS OF SEARCHABLE AREAS.....	104
FIGURE 3.22. RELATIVE NUMBERS OF SEARCHABLE AREAS	105
FIGURE 4.1. SIMPLE INVENTORY MODEL THAT PERMITS BACKORDERS AND EXHIBITS BOTH STOCHASTIC DEMAND AND LEAD TIME.....	115
FIGURE 4.2. PROCESS FOR OPTIMIZING THE SIMULATED INVENTORY SYSTEM.....	117
FIGURE 4.3. SOME AI-BASED SEARCH TECHNIQUES	119
FIGURE 4.4. BEST-FIRST SEARCH ALGORITHM PSEUDOCODE	119
FIGURE 4.5A. A NETWORK REPRESENTATION OF FIVE CITIES.....	120
FIGURE 4.5B. THE SEARCH TREE CORRESPONDING TO FIGURE 4.5A	120
FIGURE 4.5. AN EXAMPLE TO ILLUSTRATE THE BEST-FIRST SEARCH PROCEDURE	120
FIGURE 4.6A. THE DOMAIN OF THE SIMULATION OPTIMIZATION PROBLEM WITH RUNS AT THE CORNERS	125
FIGURE 4.6B. ADDITIONAL RUNS MADE IN THE DIVIDE STAGE.....	125
FIGURE 4.6. USING A DIVIDE-AND-CONQUER STRATEGY TO DETERMINE DESIGN POINT SPACING	125
FIGURE 4.7. A PORTION OF THE SEARCH TREE FOR FINDING GOOD STARTING POINTS	127

FIGURE 4.8. THE BASIC STARTER ALGORITHM	130
FIGURE 4.9. A CONTOUR PLOT OF THE THEORETICAL RESPONSE SURFACE FOR EXAMPLE 1.....	135
FIGURE 4.10A. THE INITIAL (USER-SPECIFIED) SEARCH REGION \mathcal{R}_1	137
FIGURE 4.10B. THE FIRST PASS THROUGH THE BEST-FIRST SEARCH LOOP.....	137
FIGURE 4.10C. THE SECOND PASS THROUGH THE BEST-FIRST SEARCH LOOP.....	141
FIGURE 4.10D. THE THIRD PASS THROUGH THE BEST-FIRST SEARCH LOOP.....	141
FIGURE 4.10E. THE FOURTH PASS THROUGH THE BEST-FIRST SEARCH LOOP.....	144
FIGURE 4.10F. THE FIFTH PASS THROUGH THE BEST-FIRST SEARCH LOOP.....	144
FIGURE 4.10. THE BFS STARTER SOLUTION TO EXAMPLE 1.....	144
FIGURE 4.11A. GRIDPOINTS WHERE SIMULATION RUNS ARE MADE (X) DURING THE BEST-FIRST SEARCH PORTION OF THE SEARCH.....	148
FIGURE 4.11B. GRIDPOINTS WHERE SIMULATION RUNS ARE MADE (O) DURING THE FIRST PASS OF THE SAFETY NET	148
FIGURE 4.11. THE LOCATION OF SIMULATION RUNS IN EXAMPLE 1	148
FIGURE 4.12A. THE MEAN OF THE RESPONSE.....	152
FIGURE 4.12B. THE VARIANCE OF THE RESPONSE	153
FIGURE 4.12. SOME RESPONSE SURFACE CHARACTERISTICS FOR THE MODEL OF EXAMPLE 2.....	153
FIGURE 4.13A. THE INITIAL (USER-SPECIFIED) SEARCH REGION \mathcal{R}_1	154
FIGURE 4.13B. THE FIRST PASS THROUGH THE BEST-FIRST SEARCH LOOP.....	154
FIGURE 4.13C. THE SECOND PASS THROUGH THE BEST-FIRST SEARCH LOOP.....	155
FIGURE 4.13D. THE THIRD PASS THROUGH THE BEST-FIRST SEARCH LOOP.....	155
FIGURE 4.13E. THE FOURTH PASS THROUGH THE BEST-FIRST SEARCH LOOP.....	157
FIGURE 4.13F. THE FIFTH PASS THROUGH THE BEST-FIRST SEARCH LOOP.....	157
FIGURE 4.13G. THE SIXTH PASS THROUGH THE BEST-FIRST SEARCH LOOP.....	158
FIGURE 4.13. THE BFS STARTER SOLUTION TO EXAMPLE 2.....	158

FIGURE 4.14A. GRIDPOINTS WHERE SIMULATION RUNS ARE MADE (X) DURING THE BEST-FIRST SEARCH
PORTION OF THE SEARCH..... 159

FIGURE 4.14B. GRIDPOINTS WHERE SIMULATION RUNS ARE MADE (O) DURING THE FIRST PASS OF THE
SAFETY NET 159

FIGURE 4.14. THE LOCATION OF SIMULATION RUNS IN EXAMPLE 2 159

FIGURE 4.15. THE RESPONSE SURFACE TO BE OPTIMIZED IN EXAMPLE 2 161

FIGURE 5.1. THE SIMULATION-OPTIMIZATION PROCESS..... 166

FIGURE 5.2. GREENWOOD-REES-CROUCH SIMULATION-OPTIMIZATION ARCHITECTURE..... 168

FIGURE 5.3. VISUALIZATION OF THE LEARNER AND ITS ENVIRONS 169

FIGURE 5.4. AN OVERVIEW OF THE CLASSIFIER KBSOS 172

FIGURE 5.5. CROUCH’S LEARNING PROCESS (CROUCH, 1992)..... 177

FIGURE 5.6. THE FRAWLEY ET AL DISCOVERY PARADIGM..... 180

FIGURE 5.7. THE DISCOVERY LEARNER AND ITS INTERACTION WITH THE CLASSIFIER KBSOS..... 182

FIGURE 5.8. A LATTICE SHOWING THE INTERCONNECTIONS OF THE SESSIONS HISTORY DATABASE FRAMES. 182

FIGURE 5.9A. CONCEPTS FRAME 186

FIGURE 5.9B. SESSION FRAME..... 186

FIGURE 5.9C. RULE FRAME..... 186

FIGURE 5.9. EXAMPLES OF FRAMES 186

FIGURE 5.10. DETAILS OF THE DOMAIN KNOWLEDGE COMPONENT OF THE DISCOVERY LEARNER..... 190

FIGURE 5.11A. INITIAL LINK-OF-INFLUENCE..... 191

FIGURE 5.11B. SUBSEQUENT LINK-OF-INFLUENCE 191

FIGURE 5.11. LINK-OF-INFLUENCE 191

FIGURE 5.12A. FROM BELOW..... 200

FIGURE 5.12B. FROM ABOVE 200

FIGURE 5.12C. FROM WITHIN 200

FIGURE 5.12D. THE CASE INVOLVING THE FIVE RELEVANT SESSIONS IN THE SESSIONS BASE 200

FIGURE 5.12. EXAMPLES OF LINE SEARCH..... 200

List of Tables

TABLE 3.1 DEFINITION OF EXPERIMENTAL CASES	64
TABLE 5.1. A PORTION OF THE CLASSIFIER KNOWLEDGE BASE	195
TABLE 5.2. A PORTION OF THE SELECTOR KNOWLEDGE BASE	196
TABLE 5.3. A PORTION OF THE DETAILER KNOWLEDGE BASE.....	197
TABLE 5.4. SOME SESSIONS BASE DATA.....	198

Chapter One: Introduction

Simulation

Simulation has evolved from custom programs to simulation languages that assist in model development. Early custom programs required expertise in simulation as well as in programming skills. Since each simulation was a custom job, it meant “reinventing the wheel” each time. The earlier simulation languages provided procedures that were common to most simulations, such as keeping track of an event calendar. Later, improvements added the ability to develop the simulation model graphically. The trend in simulation languages has been to reduce time spent on tasks not directly related to model development.

Development and use of simulation models typically require expertise in several disciplines (statistics, numerical analysis, systems analysis). Research is underway to build programs that assist in the model development process. The apparent driving force for a lot of this work is simulation’s usefulness as an effective decision making tool, including the ability to predict system behavior under different input

conditions, and the ability to conduct “what-if” analysis. A relatively recent use of simulation optimization, the determination of a set of inputs that produces an optimal (or desired) output.

Although simulation is very good at predicting the output(s) for a given state of a system, simulation optimization is not a simple and direct extension of simulation. Rather simulation optimization has adapted search and design techniques to meet its needs. Simulation optimization techniques include response surface methodology, simulated annealing, single factor search, random search, and genetic algorithms. The application of these techniques can be very time consuming computer-intensive, and/or costly.

A primary focus of simulation optimization is deciding which search strategy/technique is appropriate and how many computer runs to invest in the effort. These decisions are often made with little information about the nature of the simulation response (called the “response surface”), including presence of multiple optima, degree of variance, and activity of different input factors. One such approach suggested by Crouch (Crouch, 1992) and expanded upon by Crouch, Greenwood, and Rees (1995) is to make computer runs in a manner that leads to a characterization of the response surface, from which the most appropriate strategy can be inferred.

In particular, Crouch (1992) formulated a knowledge-based system to guide the selection of an appropriate strategy for simulation optimization. She developed a scheme for classifying a response surface and then applying heuristics to choose the most appropriate search strategy. As the search progressed and more information about the surface became available, the knowledge-based simulation optimization system (KBSOS) reclassified the response surface and changed the search strategy accordingly. However, her approach was never tested on real simulations, but rather was used with a known mathematical function that emulated a simulation.

In this dissertation the KBSOS presented in Crouch is tested on actual simulation models. An alternative scheme for response surface classification and search strategy selection is presented, as is an alternative approach for the selection of initial points.

Crouch also presented a framework for machine learning in the context of the knowledge-based simulation optimization system. The goal of her “learner” was to improve the ability of the KBSOS to guide future optimizations. Specifically, the heuristic knowledge (rules) in the knowledge base were improved (e.g., rules were added, modified, or combined). Her learner made judgments using information from two sources. The first source was past experience -- all the information generated during previous simulation optimizations. The second source was results of experiments that the learner had performed to test hypotheses regarding KBSOS rules.

It is the framework of Crouch that serves as the blueprint here for the construction of an improved “learner.” We introduce discovery systems (Frawley et al., 1992) to take advantage of the past history that the system has diligently collected.

The next sections of this chapter present concepts and terms that are foundational to the following discussions of the knowledge-based simulation optimization system and learner. First, simulation and issues in simulation optimization are explored. Expert or knowledge-based systems, machine learning, discovery systems, and knowledge-based simulation optimization are discussed in the following four sections. Finally, the motivation for combining simulation optimization with a knowledge-based system and discovery learning is presented, and the plan for the rest of the dissertation is given.

Simulation and Simulation Optimization Issues

Simulation is commonly recognized as one of the most widely applied computer modeling techniques in use today. Its popularity is evidenced by the large number of applications documented in the literature and the extensive breadth of problem domains to which it has been applied. With the advent of rapidly advancing computer technology, the widespread use of simulation is expected to accelerate. The value of simulation is that it permits the study of systems that cannot feasibly be constructed or experimented upon in the “real world,” and which are too complex to be analytically modeled. Simulation is very useful in predicting the output of a system or its response to a given set of input conditions. However, it does not in and of itself indicate the input conditions required to achieve a desired response. Simulation is an evaluative methodology and not an optimization technique.

In many cases the strategic objective of a study is to find the best solution for the system under investigation, i.e., optimize the system’s performance. When the search for the optimal solution involves the use of data obtained from a simulation model, the analysis involves the process of *simulation optimization*. The optimization process is complicated by the presence of random error, often the result of the combined random effect of uncontrollable conditions.

Note that simulation optimization is referred to as a *process* and not as a technique, methodology, or algorithm. In fact, the process of simulation optimization typically utilizes a wide range of mathematical and statistical tools. There is no single or standard approach to optimizing a system where the data for the analysis is based on experiments conducted with a simulation model. Some approaches focus on a single simulation run (e.g., frequency-domain analysis, perturbation analysis). Others focus on a search process that involves multiple simulation runs. Within this approach, which is the most common, there are many philosophies on how the search should be conducted. For example many approaches utilize the

data to fit metamodels (e.g., response surface methodology, neural networks, nonparametric regression); others are free of underlying model assumptions (e.g., random search, Box’s complex search, genetic algorithms). Yet another approach (Crouch, Greenwood, Rees, 1995) (Greenwood, Rees, Crouch, 1993) proposes a multi-strategy process that utilizes the “best” methodology based on current experimental and synthesized knowledge of the search environment. This brief discussion of the approaches to simulation optimization is meant to illustrate the diverse and varied literature that exists to solve this difficult problem. It is beyond the scope of this section to review all of these approaches to simulation optimization. Therefore, the interested reader should refer to overview or literature review articles and introductory texts on the subject, e.g., (Azadivar, 1992) (Barton, 1992) (Jacobson, Schruben, 1989) (Meketon, 1987) (Myers, 1971) (Safizadeh, 1990).

In general, the simulation optimization problem can be expressed as:

$$\text{Optimize: } E[\mathbf{Y}] = E[f(\mathbf{X} | \mathbf{Z})] \tag{1}$$

$$\text{Subject to: } \mathbf{h}(\mathbf{X}) \leq \mathbf{0} \tag{2}$$

where the responses that are to be optimized, $\mathbf{Y} = (Y_1, Y_2, \dots, Y_m)$, are functions of controllable factors, $\mathbf{X} = (X_1, X_2, \dots, X_n)$, uncontrollable conditions, \mathbf{Z} , and random error, \mathbf{e} ; i.e., $\mathbf{Y} = E[\mathbf{Y}] + \mathbf{e} = f(\mathbf{X} | \mathbf{Z}) = E[f(\mathbf{X} | \mathbf{Z})] + \mathbf{e}$. Each response Y_j is a random variable and takes on a set of values for the same setting of the controllable factors; i.e., there is some distribution of Y_j values for each combined level of the controllable factors. To model this behavior each response is oftentimes considered equal to the sum of a constant and a noise term that represents the random error, where the constant is the expected value of the response, $E[Y_j]$, for a specific combination of factor settings. Therefore, due to the presence of random error, the optimization process typically focuses on the expected value of the responses. But, while the goal is to optimize $E[Y_j]$, only Y_j is observable. Also, each objective regarding Y_j involves either the absolute maximization (or minimization) of Y_j or the achievement of Y_j to exceed some goal by a

specified tolerance. The simulation optimization problem is constrained, at least by the bounds of the region to be explored. As shown in (2), $\mathbf{h}(\mathbf{X})$ is a vector of deterministic constraints typically of the form: $L \leq \mathbf{X} \leq U$ or $L \leq f(\mathbf{X}) \leq U$, where L and U are the lower and upper bounds of the search region, respectively. Typically the regional boundaries change as the search process progresses. For example, as more information becomes known about the search environment and characteristics of the surface, the search region narrows so as to include only the most promising sector(s). The domain of the region may be either continuous, discrete, or mixed.

Knowledge-based Systems

Expert or knowledge-based systems is a branch of artificial intelligence that has grown in prominence and application in the last ten to twenty years. Feigenbaum has defined an expert system as “an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution” (Harmon and King 1985). Expert systems are set apart from traditional computer applications in that they can: manipulate symbols (words, phrases, lists of words, etc.); reason using heuristics (“rules of thumb” developed over time by experts); function with uncertain or incomplete knowledge (traditional programs usually stop executing if needed information is unavailable); and explain how a conclusion was reached or why requested information is needed.

The benefits of expert systems are many. An expert's knowledge about his/her field of interest can be captured in an expert system, making it available to non-experts, freeing up the individual to tackle other important problems and tasks, and providing a mechanism for "keeping the knowledge alive" even after the expert leaves the firm or organization. If the application is one for which a team of experts is usually required, the expert system makes it possible to have the expertise of these different individuals available in one place, twenty-four hours per day, seven days per week. Expert systems do not have "off" days -- they do not get sick or take vacations, and they always remember everything they have learned.

These benefits address some of the issues raised in the last section. An expert system could give a non-expert access to simulation optimization expertise; this could encourage more use of simulation and simulation optimization. Also, simulation optimization expertise and research findings could be assembled in one expert system, whereas now the information is distributed in time and geographical location among many different researchers, practitioners, and publications.

Rolston (1988) describes a typical expert system architecture as having five parts, as shown in Figure 1.1. The knowledge base contains domain-specific knowledge: facts, procedural rules (well-defined rules that describe invariant sequences of events and relations), and heuristic rules (rules of thumb usually developed through years of experience which provide direction when procedural rules are not available or relevant). The inference engine retrieves knowledge from the knowledge base and infers new knowledge from it as required by the user. The explanatory facility, when asked, provides the user with explanations of how a conclusion was reached or why certain information is being requested from the user. The knowledge update facility is a mechanism for updating and/or modifying the knowledge stored in the system. Finally, the user interface connects the user to the other parts of the system. Expert systems are beginning to include another component, the program interface. This component allows expert systems to call and be called by external programs -- spreadsheets, databases, FORTRAN programs, etc. -- and greatly adds to their flexibility.

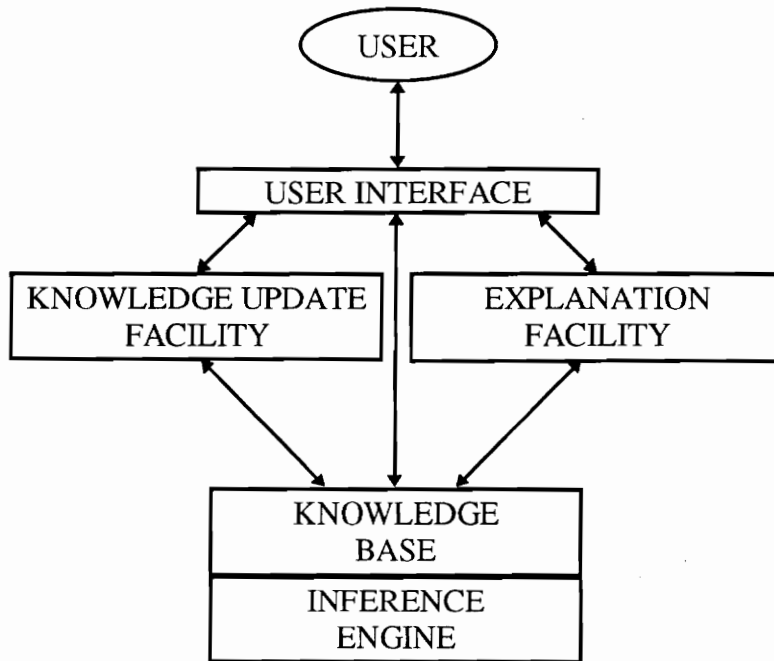


Figure 1.1. Expert system architecture

Traditionally, expert systems have been written in the artificial intelligence languages LISP and PROLOG. The complexity of the systems and the languages in which they were written restricted the broad development, and therefore, use of expert systems. This situation has changed and continues to change dramatically since the advent of expert system shells.

An expert system shell is just what the name implies -- the shell of an expert system. Shells contain all the components of an expert system except domain-specific knowledge. Hence one shell can be used to create a variety of expert systems by varying the knowledge base on which it operates. Shells are available for mainframes, minicomputers, and personal computers, with varying levels of complexity, flexibility and cost. For this research the shell VP-Expert (1989) is used on a personal computer.

In recent years there has been a trend toward using the term "knowledge-based systems" instead of "expert systems" since not all such systems contain truly exclusive, expert-level knowledge. The terms are often used interchangeably; in this work knowledge-based systems is generally used.

Machine Learning

Although the Crouch's knowledge-based system provides guidance for carrying out simulation optimizations, it does not nor is meant to include all known search strategies or classification characteristics. These are things that can be added over time, as appropriate, via machine learning. "Machine learning" means that a computer system (the machine) improves itself over time (learns). How this can be done for simulation optimization is discussed in Crouch. Consider first why it should be done.

According to Forsyth and Rada (1986), "learning algorithms attempt to achieve one or more of the following goals: provide more accurate solutions; cover a wider range of problems; obtain answers more economically; and/or simplify codified knowledge." These goals can easily be translated into the simulation optimization context. The introduction of new search strategies or improved surface classification (which provides for more appropriate strategy choices) can result in more accurate solutions (closer to the true optimum) and more economical solutions (fewer simulation runs used to find the optimal response). Simplifying codified knowledge (i.e., the rules in the knowledge base) by removing classifications that do not contribute to strategy selection or by combining overlapping rules provides two benefits. It will streamline the knowledge base, thereby saving storage space and reducing execution time, and will increase our understanding of what information about a surface is essential to successful simulation optimization.

Knowledge Discovery and Discovery Systems

Frawley, Piatetsky-Shapiro, and Matheus (1992) present a prototypical framework for knowledge discovery under a different setting than simulation optimization, namely databases. This framework is redrawn in Figure 1.2; it contains five components (besides the discovered knowledge itself).

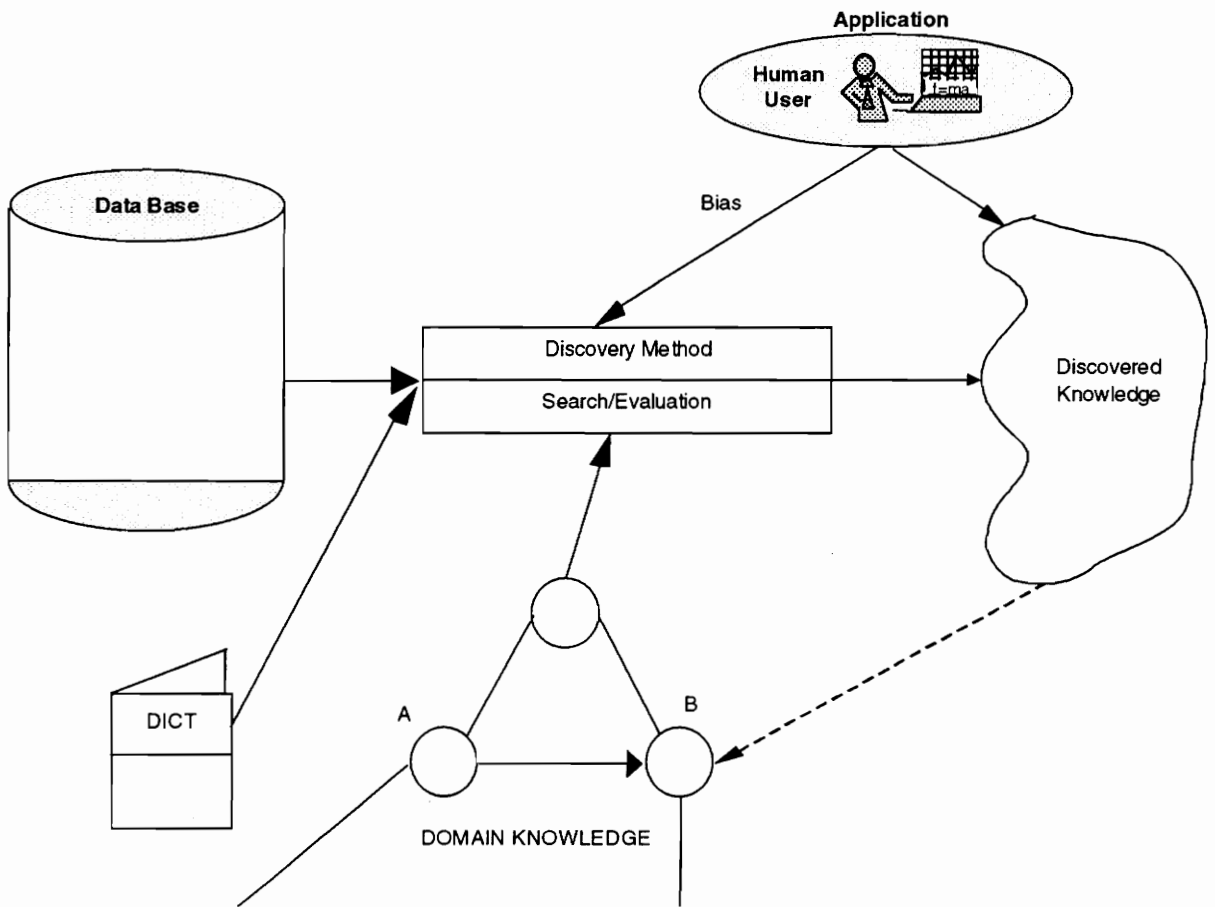


Figure 1.2. The Frawley et al. Discovery Paradigm

The Frawley discovery system has as its core the discovery method, which computes and evaluates patterns on their way to becoming knowledge. Note in Figure 1.2 that the discovery method has two

principle components: search and evaluation. Inputs to the discovery method include the database itself, its data dictionary (which defines field names, the allowable data types for field values, various constraints on field values, etc.), additional domain or background knowledge, and a set of user-defined biases that provide high-level focus. The output of the discovery method, of course, is discovered knowledge that can be directed to the user and/or fed back into the system as new domain knowledge. Frawley et al. note that both the user bias and the domain knowledge assist discovery by focusing search; i.e., these sources guide and constrain search by, for example, telling a system what to look for and where to look for it. These constraining influences are both desirable and undesirable: the former in that discovery is made easier, and the latter in that valuable discovery may be ruled out by the constraints.

Frawley et al. (1992) point out that discovery algorithms inherently contain two processes: identifying interesting patterns and then describing them in a concise and meaningful manner. They note that the identification problem is essentially a problem of pattern identification or clustering, which in essence is the problem of finding classes such that the similarity within classes is maximized while the similarity among classes is minimized. For example, it might be important for a firm to discover that the major purchaser of its product is a particular set of individuals, whereas other individuals tend to have very little interest. Concept description involves the summarization of relevant qualities of the pattern classes rather than just enumerating them. For example, it would help the firm described above to know that the particular set of individuals is the class of white males between the ages of 15 and 20. According to Frawley, well-known approaches to concept description include decision-tree inducers (Quinlan, (1986)), neural networks (Rumelhart and McClelland, (1986)), and genetic algorithms (Holland et al., (1986)).

Knowledge-based Simulation Optimization

A simulation model can be thought of as a “black box,” with controllable inputs feeding into the box, and the simulation model’s responses leaving the box as outputs. The simulation model provides an approximation of how the true system it represents would respond to the given inputs. Each response can be considered to be a function of the inputs with a random error term added.

Figure 1.3 depicts the simulation-model box together with another black box in a feedback loop around it. This second box represents the simulation optimizer. The optimizer takes outputs of the simulation model and uses them to suggest new values for the inputs to the simulation model. The objective of the optimizer is to find inputs that will result in optimal or satisfying responses from the simulation model.

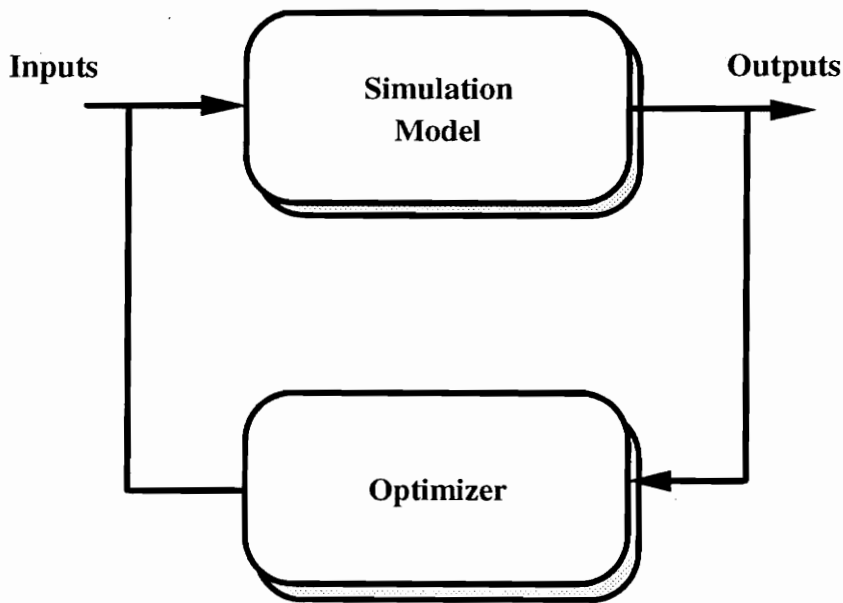


Figure 1.3. The simulation-optimization process

The need for simulation optimization and the costs involved in it have motivated the development of different strategies to search for optimal-response-producing input levels. These strategies range from

random and single-factor searches to response surface methodology (RSM) to simulated annealing and genetic algorithms. Meketon (1987) divides simulation optimization strategies into three general categories: nonlinear programming techniques, RSM, and stochastic approximation.

An important decision that must be made in simulation optimization is which search strategy to employ. Some work has been done to aid this decision, although Meketon concludes that “optimization for simulation, to date, remains an art, not a science.” He considers the information available (or assumed) about the simulation, and groups optimization methods accordingly to help narrow the choices. Safizadeh (1990) discusses a variety of strategies and their application and concludes that generally RSM approaches are most effective, although some new developments look promising. Smith (1973) performed an empirical study of the effectiveness of several search strategies (random search, single factor search, and four variations of RSM) on a variety of surfaces. He found that the relative effectiveness of each of the strategies varied depending on the characteristics of the response surface (presence of local optima, random error, number of controllable inputs, etc.).

Surveys of simulation optimization lead to the conclusion that organized guidance is needed to help users choose appropriate search strategies. Safizadeh (1990) explains that: “for successful design and analysis of simulation, one should be well versed in several disciplines.” Because of this, users are inhibited from using simulation optimization (and thereby simulation). He concludes that there is, therefore, a need to “develop interactive programs that direct a user to an appropriate optimization technique.”

In an earlier paper regarding selection of appropriate optimization technique, Greenwood, Rees, and Crouch (1993) pointed out that there is both art and science in simulation optimization. They further suggested that the art and science should be “separated” in a simulation optimizer, and, in particular, that procedural (e.g., third generation) languages should be used to model the science part, whereas knowledge-based approaches should be used to encapsulate the heuristics that make up the art portion. The particular architecture suggested consists of an inference engine, a knowledge kernel, and processing

support modules (see Figure 1.4). The knowledge kernel, in turn, contains three parts: a database to store results, a methodology base to store procedures, and a rule base to store heuristics and to provide control. Note that with this architecture, the fact that optimizer control is resident in the rule base implies that there is no set algorithm for simulation optimization; rather the inference engine (using, for example, backward chaining) can pursue a goal using whatever rules are in the knowledge base. This implies that if the rules are or can be changed, then, in essence, the optimization algorithm itself can change. Exploiting this notion, Greenwood et al. suggested that if results are stored in a database, and if “the algorithm” can be changed by changing rules, then the potential for “doing better” next time, i.e., “learning,” exists. This notion of a learner is shown in Figure 1.5. The basic idea is that historical observations are taken from the database in the knowledge kernel of the optimizer, processed by the learner, and then rules are either added, deleted, or changed back in the optimizer rule base. In this manner, not only can heuristics be modified and improved, but so can control of the entire system.

Purpose of Research

Although Crouch specified a classifier for strategy selection in the knowledge kernel, the system was demonstrated with a function rather than a simulation. Also, the method presented for determining the mapping between response surfaces and search strategies was limited to a few surface characteristics. Not all of the types of learning presented by Crouch were demonstrated, rather some were left for future study. It is these limitations that are now addressed.

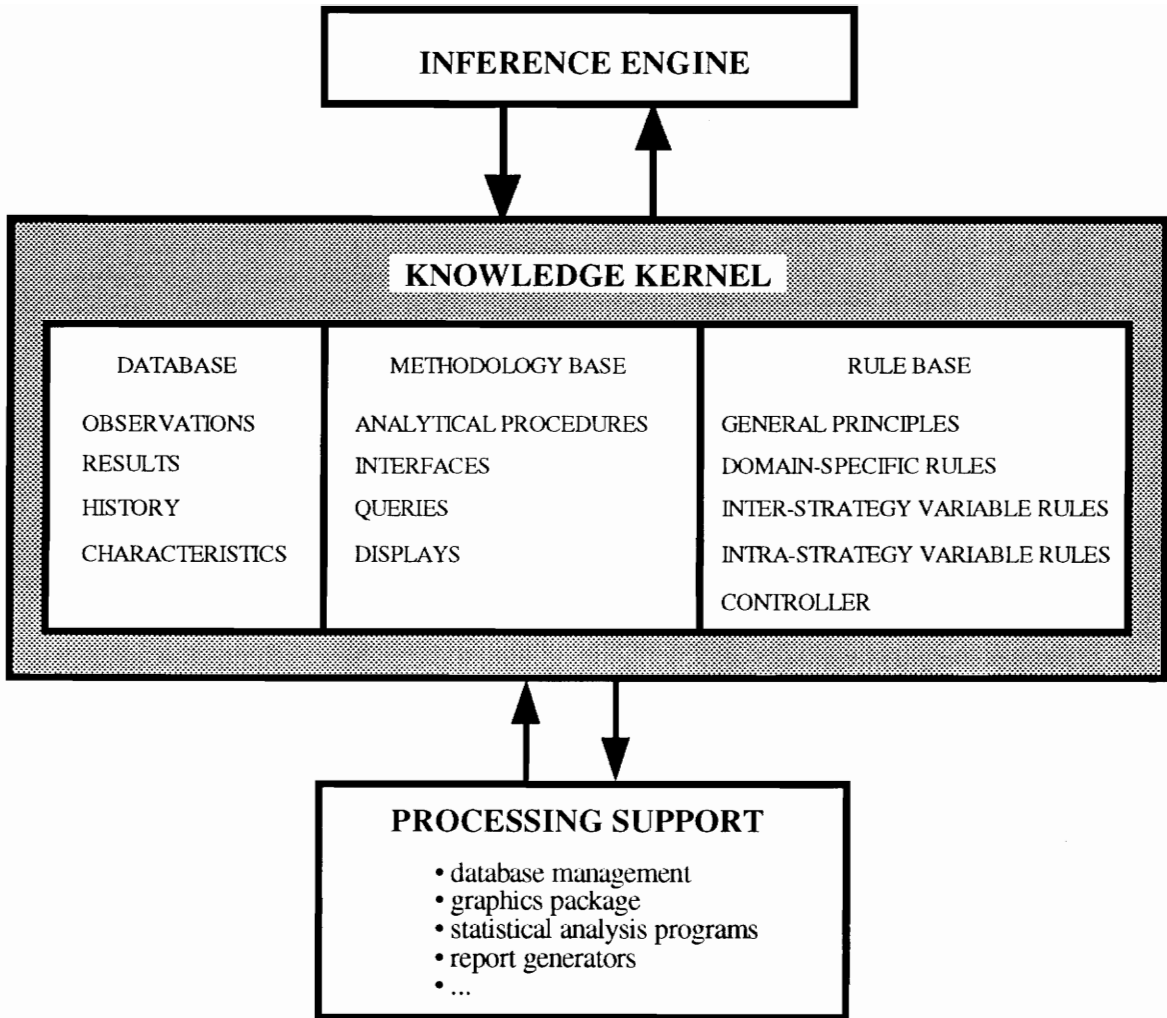


Figure 1.4. Greenwood-Rees-Crouch simulation-optimization architecture

In particular, this research tests the classifier approach as suggested in Crouch on actual simulation models instead of a function. The behavior of surfaces under different conditions is also studied to refine the method presented in Crouch. An alternative method for initiating the process is presented.

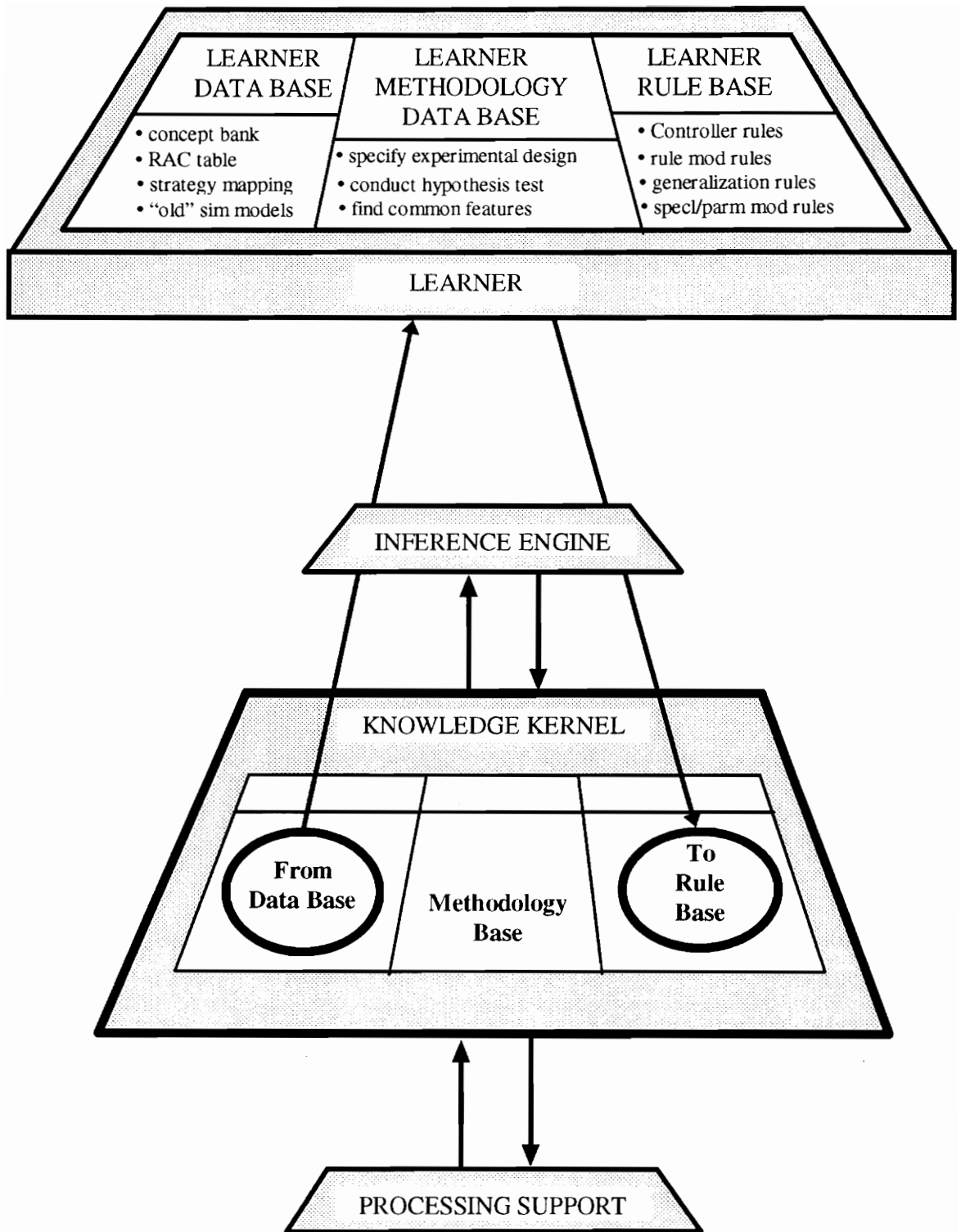


Figure 1.5. Visualization of the Learner and its environs

The first contribution of this research is the study of surface characteristics and their impact on search procedures. This provides some insight into the simulation optimization process. This dissertation lays some groundwork by examining the behavior of simulation response surfaces themselves. In particular, a simple, inventory-simulation model is studied under various experimental conditions; both point and region estimates of surface characteristics are determined and graphed while such factors as number of replications, simulation run length, and demand and lead-time variances are varied.

It is found, for example, that even for this simple surface, such optimization techniques as first-order Response Surface Methodology (RSM) are inappropriate on anywhere from 21% to 98% of the feasible region, depending on the case. Three implications are noted: the need for a simulation-optimization starter; the importance of examining global, nonparametric-metamodeling approaches to simulation optimization; and the desirability of investigating a multi-strategy approach to optimization. The first major section of this dissertation concludes with a call for further research investigating all three suggestions.

The second contribution here is the development of an alternative to Crouch's shotgun procedure for selecting initial inspection points. This is a direct result of the findings of the surface characteristics study. Many simulation optimization approaches assume that a "good" starting point is identified, that the design grid (i.e., spacing of runs for searching) is known, and that only one basic search method need be employed. Often, however, one or more of these items is unknown or is inappropriate. These assumptions can lead to an unnecessary expenditure of simulation runs, failure to find the simulation optimum, and/or a false declaration of the optimal conditions.

It is proposed here that an approach based on best-first search be used to determine the optimization starting region, starting point, and design grid. Other key features of the method that work in concert with the best-first search are a divide-and-conquer strategy for partitioning the search space and a safety

net which acts as a conservative check to prevent permanent pruning of desirable regions. The methodology is demonstrated and shown to be successful on an example problem.

The third contribution is the application of discovery learning concepts to the knowledge-based simulation optimization system of Crouch. The ideas generated are tested on simulations instead of a mathematical function

Scope and Limitations

Although significant, each of the three contributions outlined above has a limited scope. The first contribution, what we call here the surface characteristics study, is based on a single class of simulation models (inventory models), and hence is not completely generalizable. Moreover, the assumptions of each statistical test are rigidly enforced; in some cases the tests may be robust to their assumptions. This has been ignored here and should be studied in further work.

The second contribution, the best-first search starter, assumes that a first-order metamodel is fit in each region of the surface. Additional research should be conducted on alternatives to first-order metamodels such as second-order models and nonparametric metamodels. Again results presented in this dissertation have been based upon inventory models and should only be extended beyond this realm with care.

The third contribution, the introduction of knowledge discovery in a learner, is developed for only one type (parameter modification) of learning. The approach designed here pertains to other types of learning

as well, but is not directly extensible to those. Additional research is needed to pursue these other kinds of learning.

Plan of Presentation

The next chapter surveys related literature, and especially delves more deeply into the work by Greenwood, Rees and Crouch. Chapter three presents the surface characteristics study and investigates the effect of surface behavior on search methods. Chapter four outlines the best first search starter and tests it on a simulation response surface. Chapter five describes how discovery system concepts are applied in the knowledge-based simulation optimization field and demonstrates this on one of Crouch's types of learning. Chapter six summarizes contributions and presents a plan for furthering the state of the art in a learning knowledge-based simulation optimization system.

Chapter Two: Literature Review

Simulation

The development of simulation did not necessarily start out with the objective of making it a widely accepted tool. The initial attempts were perhaps focused on providing some means of analysis for problems that did not permit closed-form analysis. The improvements were geared to making it more accurate and reliable than on ease of use. The cost of computing would be a problem until the rise of the micro (personal) computer. Once simulation became a tool more businesses could afford, it became necessary to make simulation programs easier to use. This would include efforts to take care of the entire process from model development to model refinement. It is at this point where Artificial Intelligence concepts started to be applied.

One of the earliest simulation programs was developed by Tocher in the late 1950's (Tocher, 1966). Tocher also authored one of the first texts (Tocher, 1962). Other languages that followed were GPSS at IBM and SIMSCRIPT at RAND. The computers of the period had short word lengths which made it

difficult to obtain accurate numeric results. At this stage it was possible to perform simulations on a computer but the skills and costs involved prohibited wide-spread use.

The statistical aspects of simulation were the focus of the next stage in the development of simulation. Routines were developed for procedural languages like FORTRAN (Pritsker, Kiviat, 1969) to improve on random number generators and implement multiple replications. As computers became more powerful, numerical accuracy improved, and thus statistical refinements continued. These improvements would be necessary for wider acceptance of simulation as an analytical tool, but the requirement of significant programming skills was still a problem.

The next shift in focus was to tools that reduced the programming burden. These tools were called code generators and would ask the user questions about the simulation to be developed and would then aid in producing the simulation program (Mathewson, 1984). It became possible for the user to think of the simulation in terms of diagrams which could subsequently be translated into code by the simulation program. An example of this is Q-GERT (Pritsker, Sigal 1983). The work done in this area made it possible for non-programmers to develop simulation programs. However, analysis of results still needed improvement since the typical output tended to be voluminous and at times even cryptic.

Simulation programs became available on a large scale as micro computers became cheaper and faster and more powerful. Many features of the original mainframe versions of simulation programs were also included. Simple (by today's standards) animation was one of these. It was one of the improvements that helped in the analysis of output and in model validation, thereby making simulation programs accessible to more people. As the users of simulation programs increased so did the need for better tools for developing simulation models. There was also an increase in the complexity of the systems being modeled.

The response to these needs came in several forms, most of which have their roots in artificial intelligence. “Intelligent front ends” became the name of a class of software that generated the code needed to run simulation models of interest to a user. One such intelligent front end was developed for SLAM (Stanwood, Waller, Marr, 1986), some knowledge of SLAM was necessary in order to make use of the intelligent front end, in which machine learning concepts were used to generate a simulation model based on a representation of the actual system. In another intelligent front end, Quinlan’s (1979) ID3 algorithm (see e.g., TRANS (O’Keefe, 1986) represented conditional events as rules from examples rather than having the user develop the rules alone. Advisory systems (a type of expert system) would then extract in an interview with the user as much information about the particular system to be modeled. Based on the interview a set of experiments would be recommended. Stated again, the motivation was to relieve the user of many burdens in the process of developing a simulation model.

Simulation remains primarily a descriptive tool rather than an analytical one. The typical scenario involves specifying a set of input parameters and then observing the results. But there are often economic (and other) reasons for finding a set of inputs that optimizes a particular output; e.g., one might want to know the number of tellers to keep on duty in order to minimize the waiting time of customers. This need has produced an area of study called simulation optimization.

Simulation Optimization

Simulation is a widely-used computer modeling technique that has been applied to a broad scope of problems, ranging from traffic-flow analysis to job-shop scheduling to military-campaign planning. Simulation permits the study of systems which cannot feasibly be constructed or experimented upon in the “real world,” and which are too complex to be analytically modeled. When a given set of input conditions is applied to a simulation model, the model’s output, referred to as a response, provides an estimate of how the true system would respond to those inputs. Although simulation is very useful in predicting the output of a system or responses, it does not in and of itself indicate the input conditions required to achieve a desired response; i.e., it is not an optimization technique, it is an evaluative methodology. The process of finding the input conditions that yield the optimal (or near optimal) system response(s) is referred to as simulation optimization, which can be a very expensive and time consuming activity. In other words, simulation evaluations address “what if” questions by providing performance measures for a given set of input conditions, whereas simulation optimization extends the evaluations to consider “what’s best” by seeking optimum values for the input conditions.

The objective of simulation optimization is to determine the values of the input conditions, n controllable factors or decision variables, that optimize m responses, subject to a set of uncontrollable conditions (conditions that affect outcomes but are not under the influence of the decision maker). This process is complicated by the presence of random error, often the result of combined random effects of all of the uncontrollable conditions. This causes a response Y_j to become a random variable and take on a set of values for the same setting of the controllable factors; i.e., there is some distribution of Y_j values for each combined level of the controllable factors. To model this behavior each response is oftentimes considered equal to the sum of a constant and a noise term, where the constant is the expected value of the response $E[Y_j]$ for a specific combination of factor settings, and the noise term represents the random error. Due to

the presence of random error, the optimization process typically focuses on the expected value of the responses; however, while the goal is to optimize $E[Y_j]$, only Y_j is observable. Jacobson and Schruben (1989) note simulation optimization is in the class of stochastic optimization problems where the objective functions are stochastic functions of deterministic decision variables; these problems are known to be difficult to solve.

Azadivar (1992) points out that although the most common goal in simulation optimization is to optimize expected value, the goal may also involve such considerations as minimizing the risk of exceeding a threshold, minimizing dispersion, etc. Meketon (1987) refers to two classes of objectives of optimization procedures: min/max and level crossing (or root finding). The latter is of the form: find $X \ni E[Y(X)] = p$; for example, find the service rate such that customers wait more than 3 minutes 5% of the time. Meketon also indicates that the level-crossing problem is the same as the min/max problem, e.g., $\min E[(Y(X) - p)^2]$, if $\text{Var}[Y(X)]$ is constant.

In general, the responses, $\mathbf{Y} = (Y_1, Y_2, \dots, Y_m)$, are functions of the controllable factors, $\mathbf{X} = (X_1, X_2, \dots, X_n)$, uncontrollable conditions, \mathbf{Z} , and random error, ε ; i.e.,

$$\mathbf{Y} = E[\mathbf{Y}] + \varepsilon = f(\mathbf{X} | \mathbf{Z}) = E[f(\mathbf{X} | \mathbf{Z})] + \varepsilon.$$

Note that the additive error considered above is only one possible model, with $E[\varepsilon_i] = 0$, and $\text{Var}[\varepsilon_i] < +\infty$.

In addition to the above goal, the optimization will be subject to upper and lower limits on the controllable factors or some function of a combination of them. Therefore, the general simulation optimization problem may be stated as:

$$\text{Optimize: } E[\mathbf{Y}] = E[f(\mathbf{X} | \mathbf{Z})] \text{ over the region } S \subset \mathfrak{R}^n \quad (1)$$

where the domain of S may be either continuous (\mathfrak{R}_C), or discrete (\mathfrak{R}_D), or mixed,

and $\mathbf{X} = (X_1, X_2, \dots, X_n) \in S$

$$\text{Subject to: } \mathbf{h}(\mathbf{X}) \leq \mathbf{0} \quad (2)$$

where $\mathbf{h}(\mathbf{X})$ is a vector of deterministic constraints typically of the form:

$$l_i < X_i < u_i \quad i = 1, \dots, n \quad (2a)$$

$$l_{n+q} < f(\mathbf{X}) < u_{n+q} \quad q = 1, \dots, b \quad (2b)$$

where b is the number of constraints involving more than one controllable factor.

Typical Assumptions

Not all simulation optimization methods search the region S directly. For example, frequency domain methods transform the optimization problem into the frequency domain (Safizadeh, 1990), and many so-called *intrusive* procedures are single-simulation-run optimization methods (Wilson, 1987). However, a broad set of simulation optimization methods do explicitly perform a search directly over the region S . For example, different varieties of Response Surface Methodology (RSM, see Box and Wilson (1951) or Myers (1971)) assume a starting point in S then use first-order and/or second-order metamodels to suggest preferred directions of search or optimality locations. The research described in chapter four is most applicable to simulation optimization methods that search the region S directly, such as RSM, random search, and Box's complex search (Safizadeh, 1990), although any optimization approach that benefits from a carefully chosen initial region and/or requires a specification of problem granularity (see below) is a candidate for the procedures defined in this research.

Methods directly searching a region S typically make several assumptions. These often include the assumption that either a "good" starting point is known or that the choice of a starting point is unimportant to the solution of the problem. Sometimes this difficulty is obviated by selecting several starting points, solving the problem for each starting point, and selecting the most-preferred answer. Another assumption commonly invoked is that problem granularity, i.e., an appropriate grid spacing/step size, is known. For example, in using first-order RSM models, a factorial design is often utilized to

determine the direction of steepest ascent. But there is no *a priori* rationale to determine the coding of natural variables in S , i.e., to specify the size of region over which the factorial design is defined. Moreover, once a direction of steepest ascent is determined from the RSM metamodel, there again is no *a priori* reasoning that leads to a good choice of step size along the path of steepest ascent. Finally, most approaches to simulation optimization invoke only one search method throughout the entire procedure, although some have suggested hybrid approaches (Crouch, Greenwood, and Rees 1995). Sometimes a basic method is employed (e.g., RSM) with variations (e.g., first-order, second-order) to successfully address simulation models with differing amounts of curvature and/or variance in the response surface.

To summarize so far, many simulation optimization methods assume that a “good” starting point is identified, that the design grid (i.e., how far apart to space runs) is known, and that one basic search method need be employed, all regardless of the surface. Often, such assumptions are valid, for often a user has experience with the simulation model or is willing to live with the results obtained from assumptions, or expertise may be available to suggest appropriate search methods, step sizes, etc., early in the optimization process. Also, the surface may be “simple” and “smooth enough” to be impervious to the consequences of the aforementioned assumptions. However, there are cases where the simulation response surfaces are complex and have great variability in response across the surface and where little relevant optimization expertise is available. Ignoring these conditions can lead to an unnecessary expenditure of simulation runs, failure to find the simulation optimum, and/or a false declaration of the optimal conditions. Sometimes financial implications are significant. Chapter three deals with this latter class of problems where making these assumptions is not wise.

Knowledge-based Simulation Optimization

A simulation model can be thought of as a “black box,” with controllable inputs feeding into the box, and the simulation model’s responses leaving the box as outputs. The simulation model provides an

approximation of how the true system it represents would respond to the given inputs. Each response can be considered to be a function of the inputs with a random error term added.

Figure 2.1 depicts the simulation-model box together with another black box in a feedback loop around it. This second box represents the simulation optimizer. The optimizer takes outputs of the simulation model and uses them to suggest new values for the inputs to the simulation model. The objective of the optimizer is to find inputs that will result in optimal or satisfying responses from the simulation model.

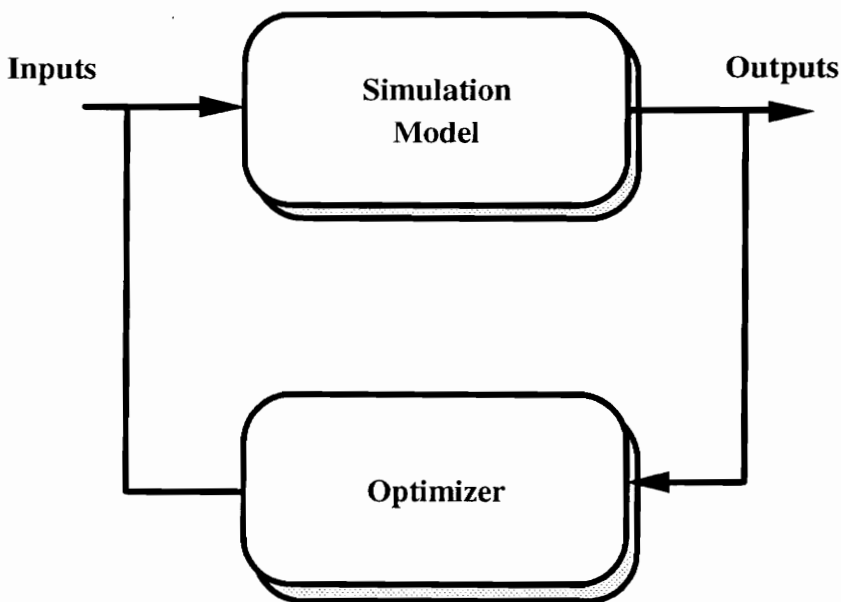


Figure 2.1. The simulation-optimization process

The need for simulation optimization and the costs involved in it have motivated the development of different strategies to search for optimal-response-producing input levels. These strategies range from random and single-factor searches to response surface methodology (RSM) to simulated annealing and genetic algorithms. Meketon (1987) divides simulation optimization strategies into three general categories: nonlinear programming techniques, RSM, and stochastic approximation.

An important decision that must be made in simulation optimization is which search strategy to employ. Some work has been done to aid this decision, although Meketon concludes that “optimization for simulation, to date, remains an art, not a science.” He considers the information available (or assumed) about the simulation, and groups optimization methods accordingly to help narrow the choices. Safizadeh (1990) discusses a variety of strategies and their application and concludes that generally RSM approaches are most effective, although some new developments look promising. Smith (1973) performed an empirical study of the effectiveness of several search strategies (random search, single factor search, and four variations of RSM) on a variety of surfaces. He found that the relative effectiveness of each of the strategies varied depending on the characteristics of the response surface (presence of local optima, random error, number of controllable inputs, etc.).

Surveys of simulation optimization lead to the conclusion that organized guidance is needed to help users choose appropriate search strategies. Safizadeh (1990) explains that: “for successful design and analysis of simulation, one should be well versed in several disciplines.” Because of this, users are inhibited from using simulation optimization (and thereby simulation). He concludes that there is, therefore, a need to “develop interactive programs which direct a user to an appropriate optimization technique.”

In an earlier paper regarding selection of appropriate optimization technique, Greenwood, Rees, and Crouch (1993) pointed out that there is both art and science in simulation optimization. They further suggested that the art and science should be “separated” in a simulation optimizer, and, in particular, that procedural (e.g., third generation) languages should be used to model the science part, whereas knowledge-based approaches should be used to encapsulate the heuristics that make up the art portion. The particular architecture suggested consists of an inference engine, a knowledge kernel, and processing support modules (see figure 2.2). The knowledge kernel, in turn, contains three parts: a database to store results, a methodology base to store procedures, and a rule base to store heuristics and to provide control. Note that with this architecture, the fact that optimizer control is resident in the rule base implies that

there is no set algorithm for simulation optimization; rather the inference engine (using, for example, backward chaining) can pursue a goal using whatever rules are in the knowledge base. This implies that if the rules are or can be changed, then, in essence, the optimization algorithm itself can change. Exploiting this notion, Greenwood et al. suggested that if results are stored in a database, and if “the algorithm” can be changed by changing rules, then the potential for “doing better” next time, i.e., “learning,” exists.

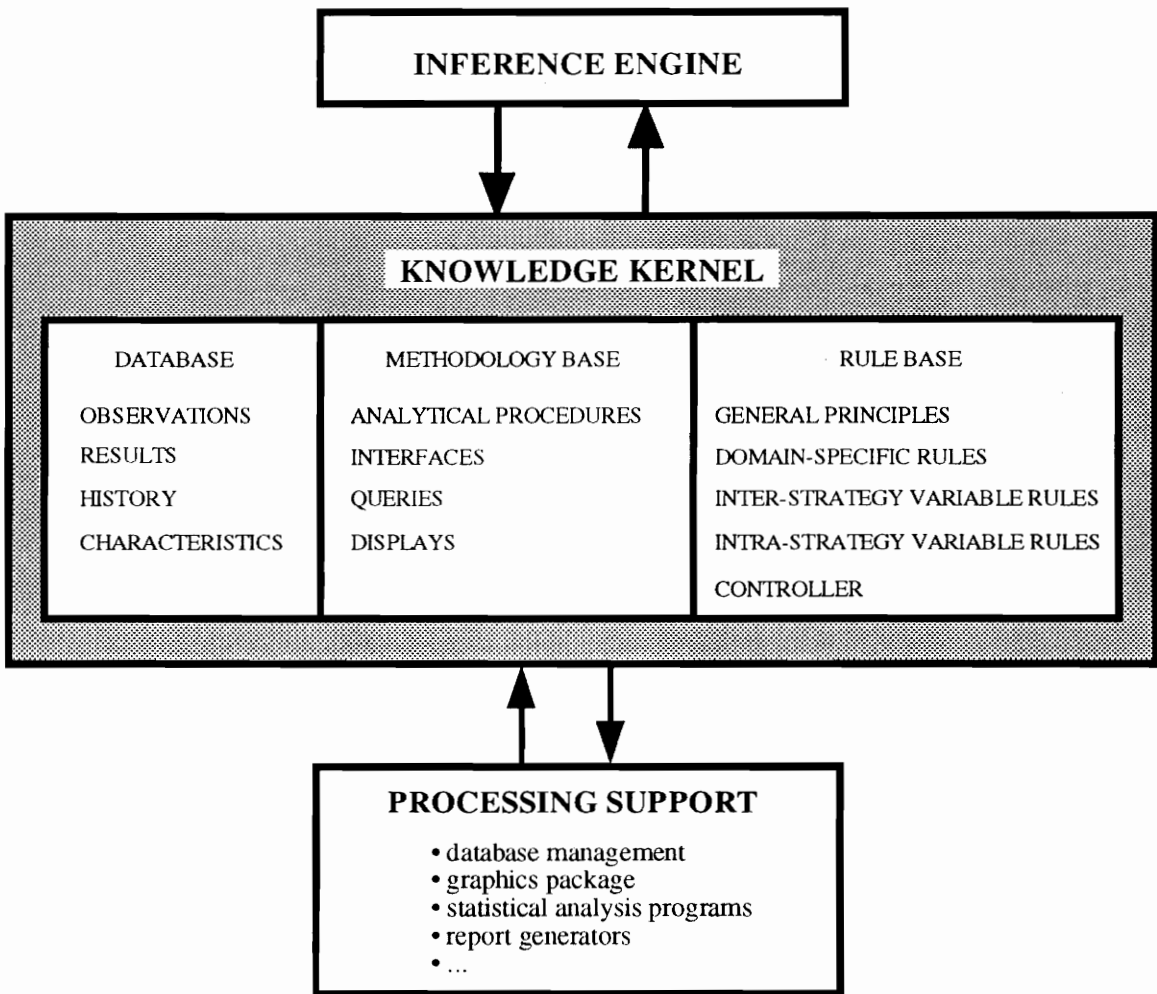


Figure 2.2. Greenwood-Rees-Crouch simulation-optimization architecture

This notion of a learner is shown in figure 2.3. The basic idea is that historical observations are taken from the database in the knowledge kernel of the optimizer, processed by the learner, and then rules are either added, deleted, or changed back in the optimizer rule base. In this manner, not only can heuristics be modified and improved, but so can control of the entire system.

Learning: Definitions, Advantages, and What There is to Learn

Crouch (1992) states that definitions by Simon and Michalski are closest to what she means when she says she will let her optimizer learn. Simon (1983) concludes: "Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same or different tasks drawn from the same population more effectively the next time." Michalski (1986) points out that knowledge acquisition seems to be the essence of most learning acts. He adds that in order to acquire knowledge, one has to represent this knowledge in some form. Consequently, he characterizes learning as "constructing or modifying representations of what is being experienced." Thus the optimizer should be able to adapt its performance so that it improves its optimization on scenarios "close" to what it has already seen. In addition, an optimizer or satisfier with a learning capability should have the capacity to modify or to construct representations of its knowledge, be it knowledge of how to reset certain parameters, knowledge that is domain specific, or knowledge that is more widely applicable as general principles.

Crouch (1992) builds upon a taxonomy developed by Carbonell et al. (1983) to suggest the types of knowledge acquisition a learner should include. The four basic types of learning are (1) rule modification or creation, (2) specialization, (3) parameter modification, and (4) generalization. According to Carbonell, specialization means adding conditions to the "if" part of a rule (the antecedent) so the rule applies to a narrower set of circumstances, and generalization means dropping restrictive conditions in the antecedent to make the rule apply in a wider variety of contexts. By parameter modification is meant the changing of a numerical value in a rule; for example, the antecedent "IF number of runs > 12" could be

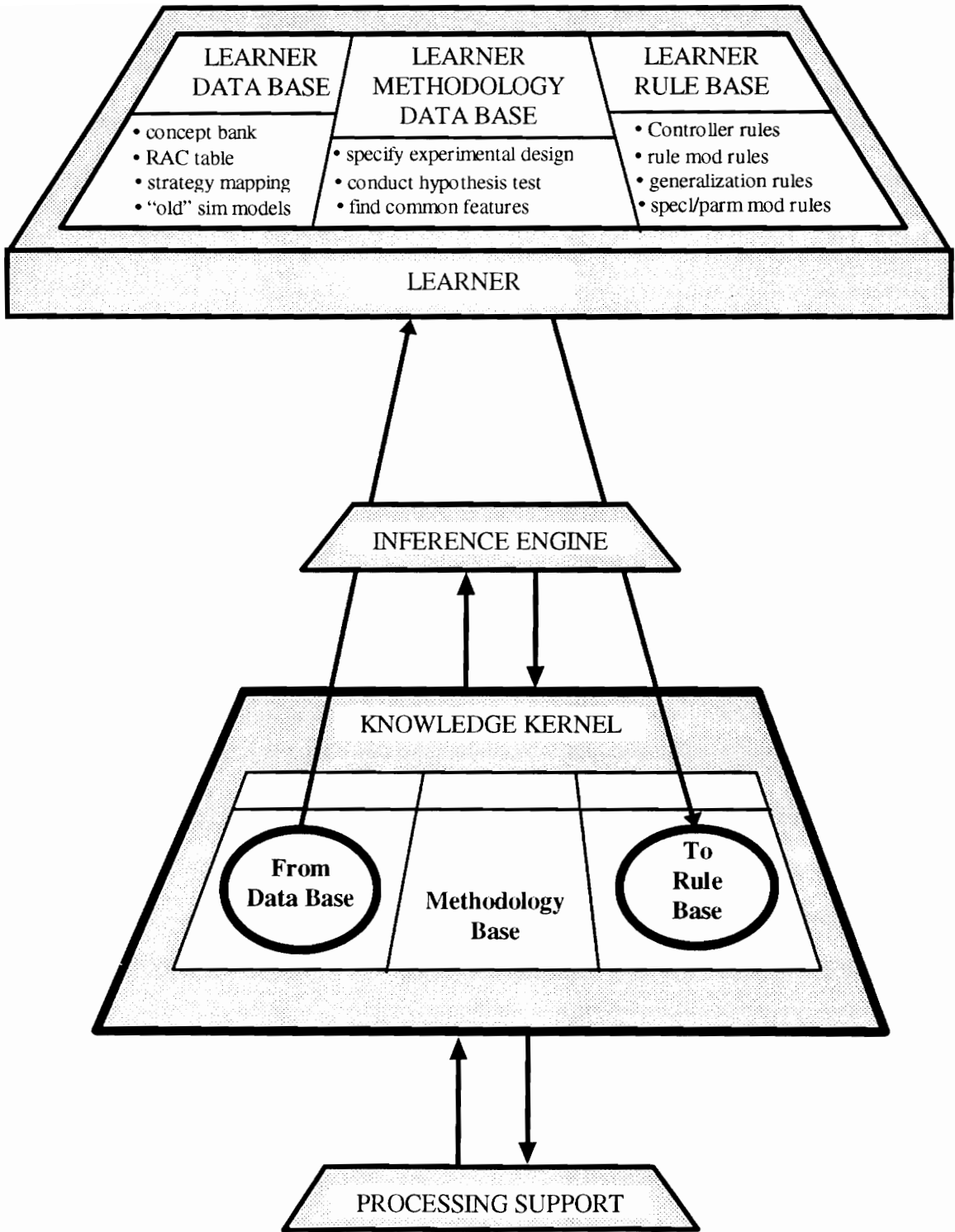


Figure 2.3. Visualization of the Learner and its environs

changed to “IF number of runs > 10.” Rule modification results in changing the consequent of a rule. For instance, a current rule may conclude that RSM is the preferred search strategy (“...THEN strategy = RSM”); however, learning may suggest that simulated annealing is preferred. Thus, the modified rule would have the consequent “THEN strategy = simulated annealing.”

In this research, we will limit ourselves to the four types of learning just elaborated, noting that additional types of learning can be added to the Learner later if desired as plug-in modules.

What it is that can be learned in a simulation optimization system with these four types of learning has been pointed out in Crouch (1992). In order to understand these ideas, however, it is first necessary to present a quick overview of CGR’s (1995) “Classifier KBSOS.” CGR called their system a “Classifier KBSOS” because its simulation output surfaces are *classified* according to the search strategy most likely to render success.

In the Classifier KBSOS, input sufficient to define the problem is obtained from the user in the User module (see figure 2.4). This input is then fed to the Classifier module, where three steps occur. First, the “shotgun” suggests an initial set of simulation runs to be made at various input combinations across the surface. The results from these computer runs are then input to the “synthesizer,” which attempts to develop a fitted or synthesized surface through those points. (A neural network can be and was successfully used for this by Crouch et al. The reason for this synthesis is that it hopefully will save computer runs by characterizing the synthesized or estimated surface rather than depending entirely on actual runs.) Then the synthesized surface is analyzed by several procedural programs and heuristics in the “characterize” module in order to classify or characterize the response surface. The idea of classifying a surface is based on a study reported by Smith in *Operations Research* in 1973, which found that optimal search technique varies by type of surface. Crouch et al. used the same explanatory variables Smith used in his study to classify their surfaces with the Classifier KBSOS.

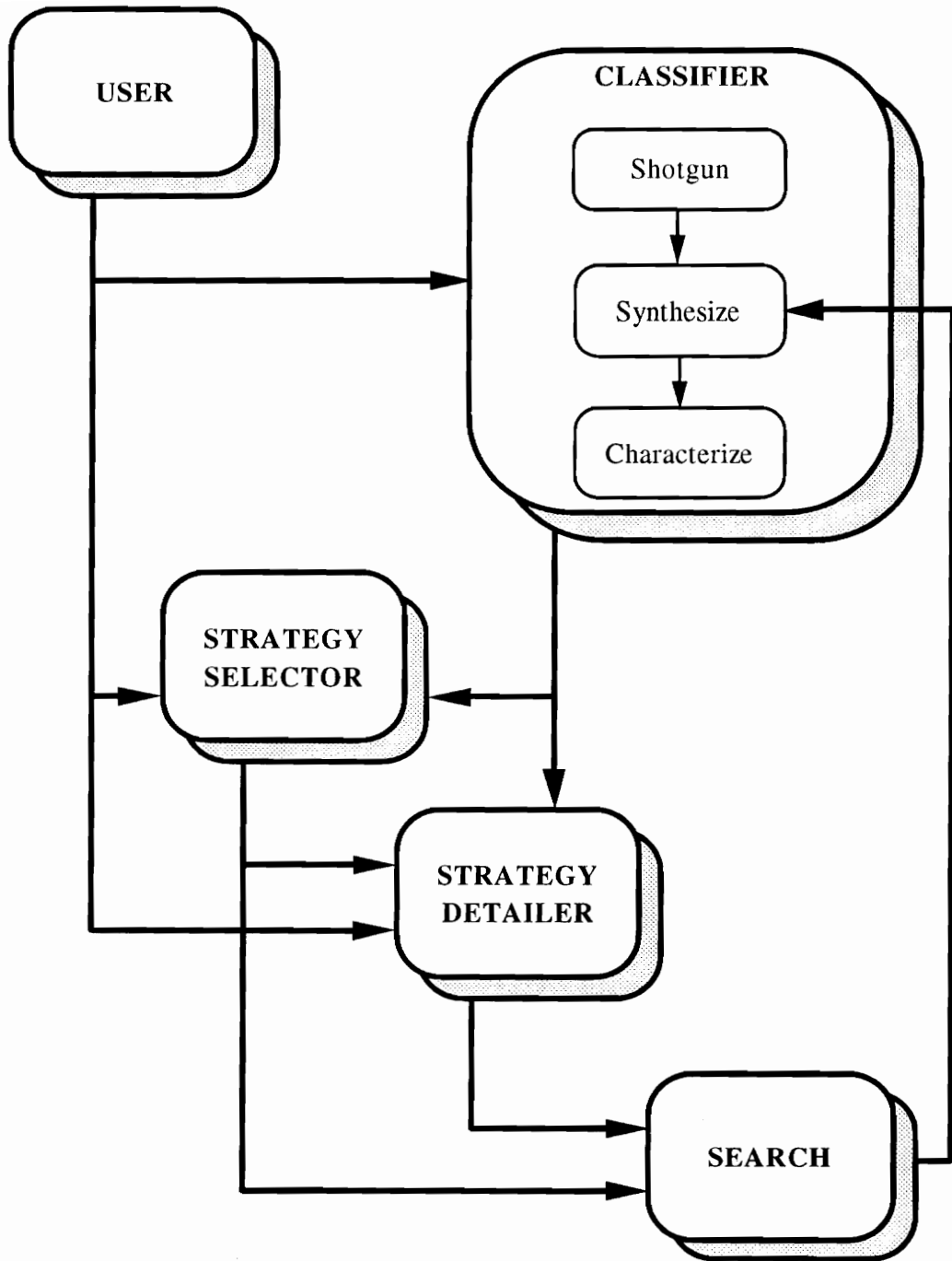


Figure 2.4. An overview of the Classifier KBSOS

Once a surface has been classified, rules in the KBSOS knowledge kernel invoke the Strategy Selector. This module is a collection of rules that choose a search strategy (e.g., RSM, random search) depending on the surface characteristics identified by the Classifier. Note that as the whole classify-and-select-strategy process is iterative, additional search may result in reclassification of the surface and hence specification of a different strategy as the optimization proceeds. After a search strategy has been chosen, the Strategy Detailer (another set of rules) is fired, and implementation particulars are set whereby the Search may be conducted.

As Crouch points out, it should be clearly stated what is not meant when one suggests that a KBSOS will learn. The learner is not expected automatically to derive or infer a never-before-seen search technique whenever a previously unanalyzed surface is encountered. Rather, the learner is expected to perform such tasks as to modify parameters in the shotgun, to suggest that a new antecedent be included in a set of rules in the Strategy Selector, or to respecify the number of runs to be made at the center point of a given search being implemented. Learning is to be incremental as opposed to far reaching, and it will only be successful as its databases of surfaces and experiments grow large.

In order to indicate how learning will take place in a KBSOS, Crouch (1992) lists some examples of each of the four kinds of learning; see that reference and Crouch, Greenwood, and Rees (1995) for further details:

parameter modification: - in the Classifier: re-specifying the number of runs to be made randomly and at regular grid points in the shotgun module; re-setting a variance threshold, above which additional replications of data points used to fit the synthesized surface will be collected; re-stipulating the vertical distance *delta* from the true optimum, within which non-adjacent portions of the response surface indicate multiple, optimal solutions. And in the Strategy Detailer, re-adjusting the step size for a given search technique.

specialization: - adding new concepts as antecedents to the rules in the Strategy Selector (e.g., adding "IF variance is not high" to a current rule specifying RSM as the search procedure); adding a similar clause again to the IF part of an existing rule in the Strategy Detailer (e.g., adding "IF lack of fit is significant" to a rule specifying a shift from a first- to a second-order RSM design).

rule modification: - in the Strategy Selector, if some cases concluding in “THEN Strategy = S_1 ” achieve different levels of success than others, then separate these cases and respecify “THEN Strategy = S_2 ,” a new strategy whereby there is some evidence that S_2 will work better on the poorer cases than S_1 did.

generalization: - deleting existing concepts from the antecedents of rules when there is evidence that such concepts are irrelevant to the Strategy Selection being made (e.g., removing “IF distance to optimum = far” from a rule concluding in “THEN Search = random search.”) Generalization is also helpful in a housecleaning sense in that rules can at times be combined, thereby reducing the number of rules in the rule base.

It is easily noted from the above lists that there are a plethora of details to be learned; this is because, fundamentally, so much of simulation optimization is heuristic, or “art.” The approach taken in Crouch (1992) and that we have taken here is to prioritize what we want to learn with our KBSOS. We have placed the Strategy Selector as our top learning objective, with its specialization, rule modification, and generalization. At second priority is the Classifier, which calls primarily for parameter modification learning.

Having examined the Classifier KBSOS, definitions of learning, and what it is that may be learned in a knowledge-based simulation optimization system, we now direct our attention to the Crouch (1992) Learner. This will provide the final building block needed to explain the Learner we have actually constructed ourselves.

The Crouch Learner

Overview: Each of the four learning types to be included in Crouch’s learner requires both procedural and heuristic computation. That is, each learning type consists of both procedural decisions such as hypothesis testing that can best be performed by algorithmic means, as well as heuristic processing best done in, for example, knowledge-based systems. A major design decision made by Crouch was to separate the “art” and “science” in the learner; Crouch, Greenwood, and Rees (1995) also did this in their KBSOS.

Figure 2.3 shows Crouch's learner sitting above the KBSOS and deriving input from the KBSOS database; changes are passed back to the KBSOS rule base. Figure 2.3 explicitly illustrates the implementation of the separation of art and science in the learner in terms of its three modules, the Learner Data Base, the Learner Methodology Base, and the Learner Rule Base. In addition, figure 2.3 shows some of the functions to be carried out by each of the three modules.

According to Crouch (1992), a knowledge-based simulation optimization system contains many concepts that may be stored in a variety of representation formats, including tables, rules, and neural networks. In order to be able to manipulate this information in a learner, the Learner Data Base must keep a registry of concepts and their interrelationships. Crouch's mechanism for doing this is a concept bank and a Relationships Among Concepts (RAC) table. The RAC table stores which concepts are used in which rules. As indicated in Figure 2.3, both the concept bank and RAC table are (important) components of the Learner Data Base, as is the strategy mapping, which will be described later in chapter five. An additional item included in Crouch's Learner Data Base is a collection of "old" simulation programs. That is, she suggested that whenever a simulation program was run and its results were stored in the database, it would be advantageous if the program (i.e., the code) itself were left in a library in the Learner Data Base, in case the Learner decided later to do further exploration with the program. Obviously, this is not practical in all cases. But the more the Learner has access to in the way of history, the more likely it is to be successful. Finally note that Crouch's Learner Data Base may share or coincide or differ from the knowledge kernel data base.

The Learner Methodology Base consists of whatever procedural aspects are necessary to implement the four types of learning. For example, if the Learner were investigating the advantages of changing a troublesome parameter, it might decide to conduct an experiment to test the proposed change. In such a case, the Learner would call the experimental design submodule, which would specify where computer runs should be made to carry out (say) a fractional factorial design. Then a second submodule in the

methodology base, a hypotheses testing procedure, would evaluate the results of these experiments to determine statistically the worth of the change. Crouch admits that these submodules may be complex, but add that they can be implemented using ideas well-established in the literature. A third submodule in the learner methodology base deals with searching for common features or concepts for a given set of rules.

Crouch's Learner Rule Base contains all the rules or heuristics needed to do specialization, rule modification, parameter modification, and generalization. Moreover, it also possesses a set of controller rules, which decide when to invoke each of the four learning types. All of these rules, under the direction of an inference engine, drive the Learner in its search for an improved simulation optimization process, and call the Learner Data Base and Methodology Base when needed.

Crouch Process Flow: A brief overview is now given of the Crouch learning process flow; details may be found in Crouch (1992). This process is based on Slade's work on case-based reasoning (1991). Slade never examined the simulation optimization context; rather Crouch adapted some of the basic concepts in case-based reasoning and learning and modified them for this application.

Figure 2.5 indicates the flow of Crouch's learning process. The shaded boxes indicate the major operations in the process needed for all four learning types. (The only exception is that Repair is not needed in Generalization learning.) The learning process for any of the types begins with Retrieve, where learner rules are used to extract relevant data from either the learner or knowledge kernel databases. Upon retrieval, learner modification rules are invoked to suggest changes in some aspect of knowledge kernel rules. This occurs in the Modify block. For example, in parameter-modification learning, a particular parameter is suggested for change; whereas in specialization learning, retrieved data cases are first segmented by performance, and concepts in the antecedents are then sought that can explain the performance differences. Once a modification is proposed that hopefully improves KBSOS performance,

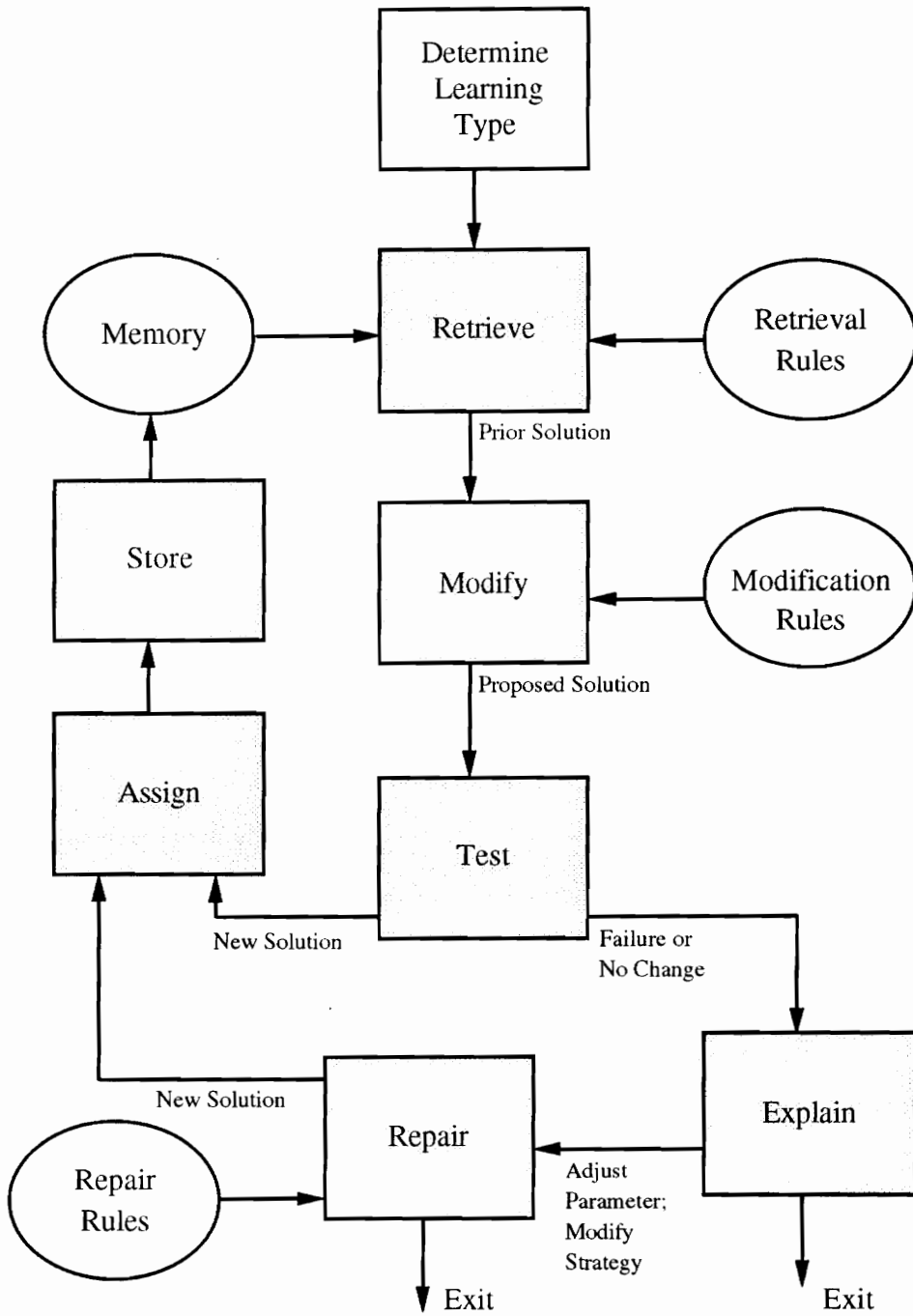


Figure 2.5. Crouch's learning process (Crouch, 1992)

the Test block is called. Basically, the Test block determines whether the proposed modification results in an improved solution (i.e., a new set of rules), or rather in no improvement or possibly failure. In the first case, control passes to the Assign and Store blocks, where the proposed modifications are actually made and put back in the KBSOS rule base. In the case of failure or no improvement, the Explain and Repair blocks are called, where either abandonment of learning for this case occurs due to unsuccessful explanation and repair, or further modification leads to a successful solution. This latter case leads back to assignment and storage, as figure 2.5 indicates.

Although Crouch's research has suggested an architecture and a learner flow, details were not specified as to how all modules would work for the four types of learning. Moreover, since a Learner has never been built, it is not known whether such a Learner is truly practical. The research described in chapter five specifically addresses these issues, making three contributions. First of all, we build a Learner and test it on a simulation example. Second, having successfully constructed a Learner, we are able to specify an architecture and process flow; in particular, it will be seen that a clear explanation of how discovery takes place was not provided in the Crouch paper. And finally, an analysis of what must be done next to extend the Learner to larger-scale, more complex scenarios is described.

The remainder of this chapter is organized as follows. The next section describes a general model of "discovery," and the following segment details the modified general learning flow of our discovery learning system. It will be found that the Crouch (1992) architecture of figure 2.3 contains most of the components necessary in a Learner, but is lacking in clear explanation of how discovery will take place -- in particular how domain knowledge and search will be used in this process. This discussion is followed in turn by a detailed inventory simulation example invoking the Learner illustrating parameter modification. The chapter concludes with a summary and a description of future steps.

KNOWLEDGE DISCOVERY

The use of knowledge discovery concepts in a knowledge-base simulation optimization system (KBSOS) is new. The previous work in KBSOS did outline and define four kinds of learning which required storing some items in the form of a database. It is a simple extension then to use knowledge extraction techniques for databases in an attempt to learn something from the data being stored.

The essence of learning as we use it here is knowledge discovery. Frawley, Piatetsky-Shapiro, and Matheus (1992) present a prototypical framework for knowledge discovery under a different setting than simulation optimization, namely databases. This framework is redrawn in figure 2.6; it contains five components (besides the discovered knowledge itself). Since our research builds a Learner based upon both the Frawley et al. paradigm and the Crouch (1992) architecture and flow, we now discuss the former in some detail.

The Frawley discovery system has as its core the discovery method, which computes and evaluates patterns on their way to becoming knowledge. Note in figure 2.6 that the discovery method has two principle components: search and evaluation. Inputs to the discovery method include the database itself, its data dictionary (which defines field names, the allowable data types for field values, various constraints on field values, etc.), additional domain or background knowledge, and a set of user-defined biases that provide high-level focus. The output of the discovery method, of course, is discovered knowledge that can be directed to the user and/or fed back into the system as new domain knowledge. Frawley et al. note that both the user bias and the domain knowledge assist discovery by focusing search; i.e., these sources guide and constrain search by, for example, telling a system what to look for and where to look for it. These constraining influences are both desirable and undesirable: the former in that discovery is made easier, and the latter in that valuable discovery may be ruled out by the constraints.

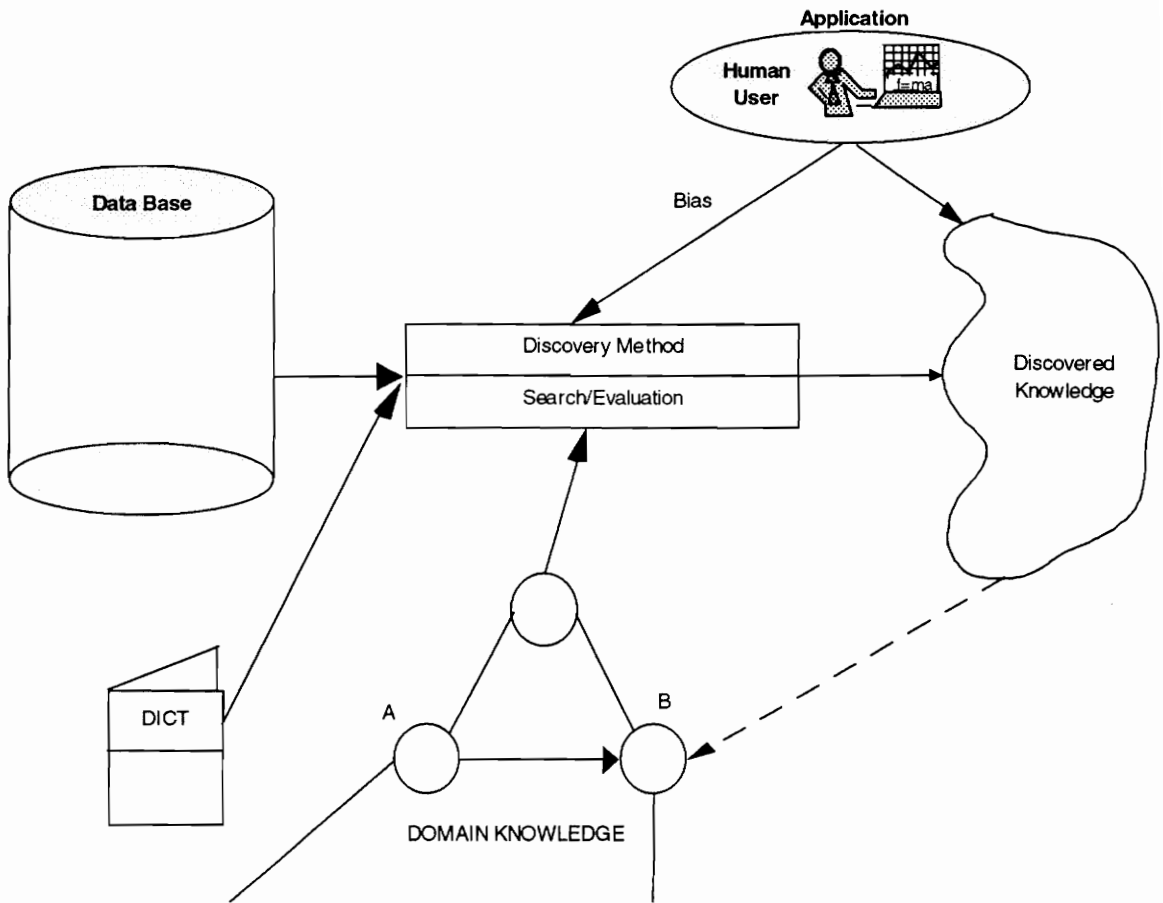


Figure 2.6. The Frawley et al. Discovery Paradigm

Frawley et al. (1992) point out that discovery algorithms inherently contain two processes: identifying interesting patterns and then describing them in a concise and meaningful manner. They note that the identification problem is essentially a problem of pattern identification or clustering, which in essence is the problem of finding classes such that the similarity within classes is maximized while the similarity among classes is minimized. For example, it might be important for a firm to discover that the major purchasers of its product is a particular set of individuals, whereas other individuals tend to have very little interest. Concept description involves the summarization of relevant qualities of the pattern classes rather than just enumerating them. For example, it would help the firm described above to know that the particular set of individuals is the class of white males between the ages of 15 and 20. According to

Frawley, well-known approaches to concept description include decision-tree inducers (Quinlan, (1986)), neural networks (Rumelhart and McClelland, (1986)), and genetic algorithms (Holland et al., (1986)).

KNOWLEDGE DISCOVERY IN THE SIMULATION OPTIMIZATION DOMAIN

Figure 2.7 illustrates the architecture of our Discovery Learner for simulation optimization and its interaction with the Classifier knowledge-based simulation optimization system. The Classifier KBSOS, shown at the right in that figure, contains three principle modules: an inference engine; a knowledge kernel, which contains the rules and algorithms necessary for simulation optimization, as well as a record (a database) of the optimization session; and processing support, including interfaces to users, the simulation program, etc. Crouch, Greenwood, and Rees (1995) may be seen for further details on the Classifier KBSOS.

The Learner, shown as an “L”-shape at the left of figure 2.7, contains the same modules as the Frawley et al. paradigm, but is adapted to fit the purposes of the simulation-optimization environment. These modules are the sessions history database, the data dictionary, a domain-knowledge module, and (at its heart), the discovery-methods module. As in Frawley, bias is provided to the Learner from a user/developer.

Note that the key information/knowledge flows between the Classifier KBSOS and the Learner consist of one primary flow from the KBSOS to the Learner, and two flows from the discovery-methods module: one back to the KBSOS, and another internal to the Learner, back to the domain-knowledge module. These three flows are emphasized in figure 2.7 by the heavier lines and arrows. The key notion is that information from optimization sessions (stored in the database of the knowledge kernel) flows to the Learner as input where it is recorded in the Sessions History Database. Similarly, what is learned by the Learner flows back as output to the rule base of the KBSOS, so that rules are modified; consequently,

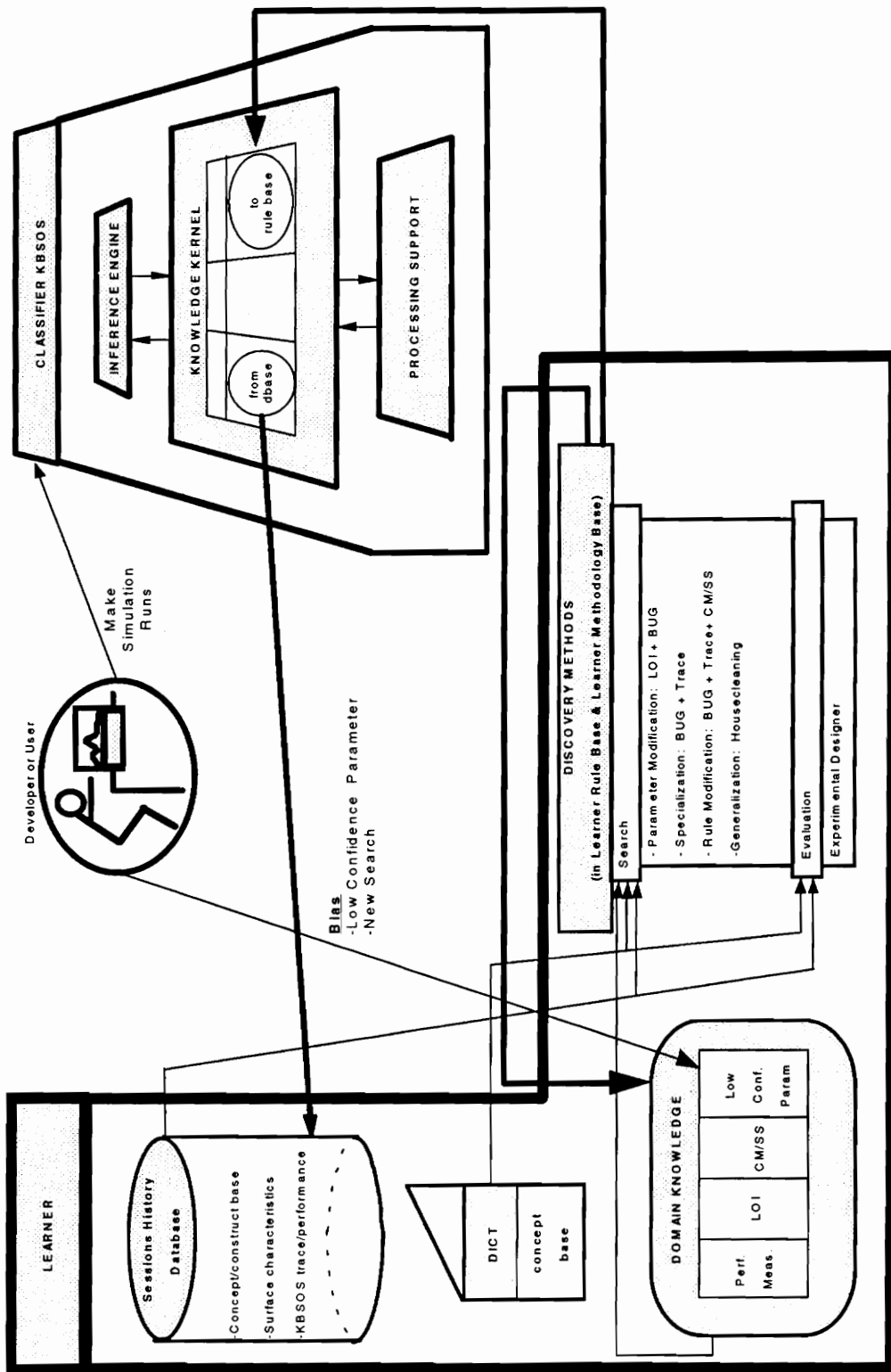


Figure 2.7. The Discovery Learner & its interaction with the Classifier KBSOS

simulations conducted in the future by the KBSOS will (hopefully) be improved. What is learned by the Learner also flows back to the domain-knowledge module in the Learner, as a means of keeping the Learner up-to-date. These flows constitute the primary activity of the Learner, with all other activities conducted in support of that activity. We now detail this support, proceeding module-by-module through the Learner.

Data Dictionary

The data dictionary maintains the concept bank, namely a list of concepts or constructs utilized in the sessions history database. For example, some of the concepts in the sessions base are number of controllable factors, distance from the optimum, level of factor activity, and presence of local optima. The concept bank also contains, as mentioned, allowable data types for field values as well as any constraints on field values. The data dictionary employed in the Learner is not significantly different from data dictionaries employed in other applications.

Sessions History Database

The Learner database is called a Sessions History database because it records the history of sessions carried out by the KBSOS. There are three kinds of information regarding any session maintained in the database, each carried to meet a different need for the Learner. The first is the concepts and the values that each can take. The second is a description of session characteristics which includes a session trace, the search method and results, surface characteristics, and the activating rule, among others. The third kind is a detailing of the rules including parameters and associated levels. Figure 2.8 is a lattice that shows some of the relationships among the three kinds of information (Siochi, 1993).

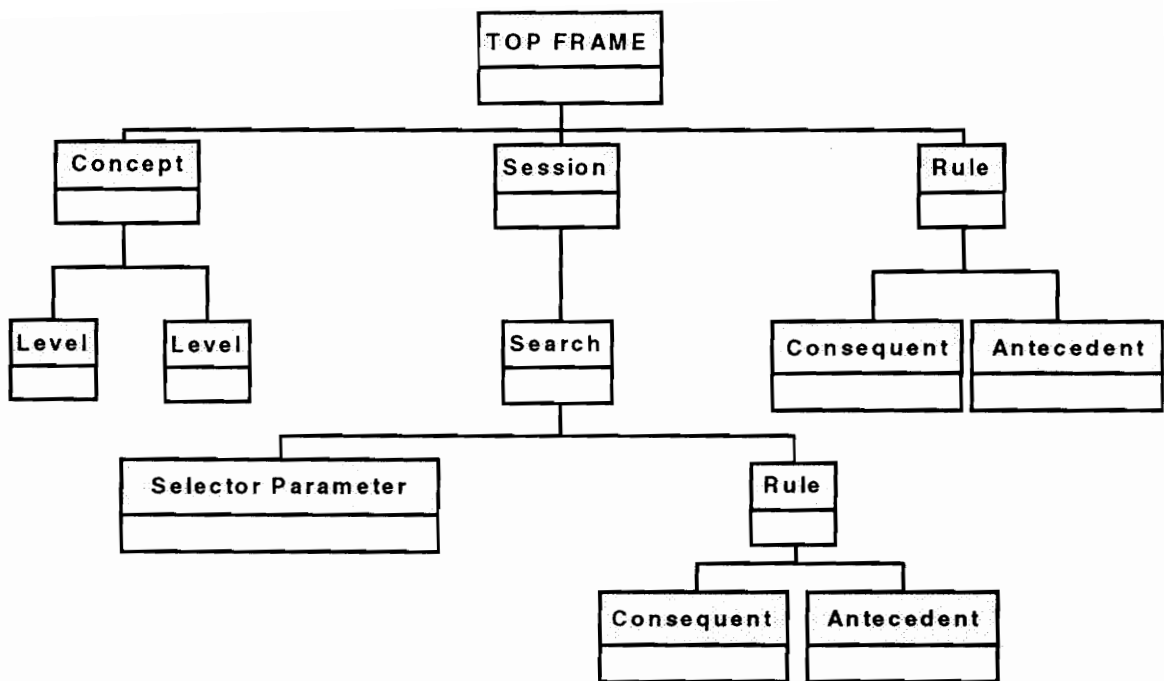
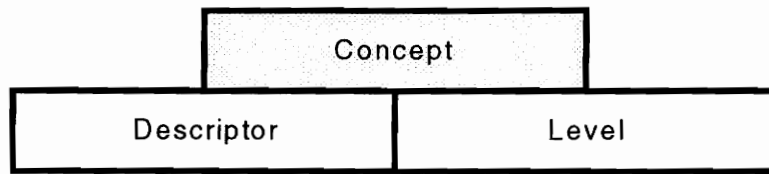
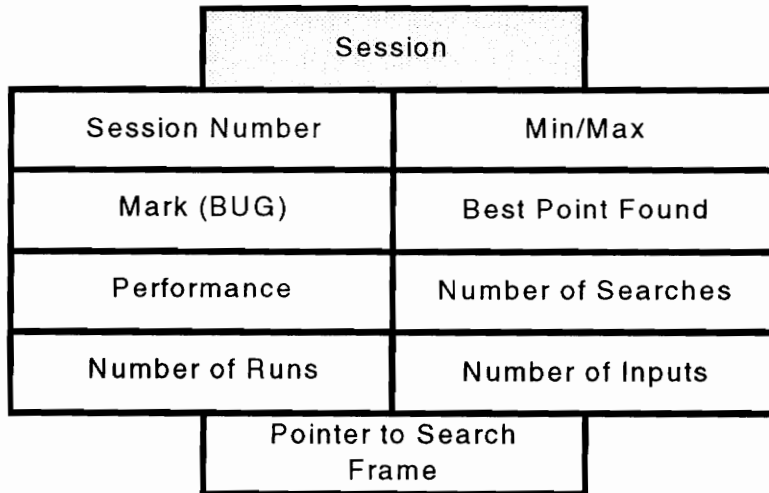


Figure 2.8. A lattice showing the interconnections of the Sessions History Database frames

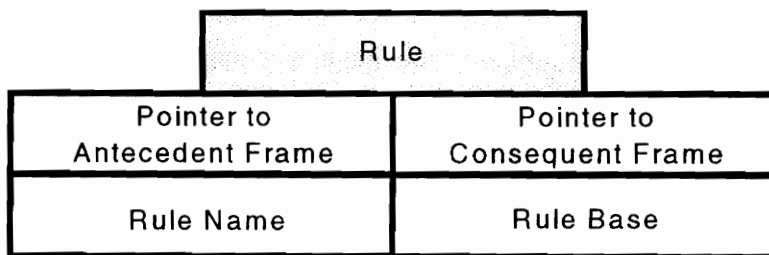
All three kinds of information are represented as frames (Siochi, 1993). The concept frame (see figure 2.9a) contains the name of the concept and the possible values that the concept can take. The concept frame can be a child frame of an antecedent or consequent frame. The session frame (figure 2.9b) contains session specific information such as the session number, the goal (min/max), performance, number of searches performed, a rating of effectiveness, total number of runs, number of inputs, and the best solution found. It has one child frame called the search frame. The search frame contains search specific information such as number of runs used, search method, best point found and the surface characteristics as estimated at that point (the selector parameter in figure 2.8). The search frame has three child frames, (1) the activating rule frame, (2) the trace frame, and (3) another search frame if an additional search had been performed (the value is null if no additional searches were performed). The rule frame (figure 2.9c) contains the rule name, the rule base it belongs to and two child frames; antecedent and consequent. The antecedent and consequent frames have pointers to concept frames and logical operator slot. A trace frame contains the points visited.



2.9a. Concepts Frame



2.9b. Session Frame



2.9c. Rule Frame

Figure 2.9. Examples of frames

Domain-Knowledge Module

The third component of the Discovery Learner is the Domain-Knowledge module. As mentioned, discovery must often be focused if the knowledge discovered is to be useful, and sometimes it must be so if there is to be any discovery at all. The general purpose of the Domain-Knowledge module is to enable the discovery that occurs in the Learner to be relevant and useful to the Classifier KBSOS. In particular, the function of the domain-knowledge component is to provide guidance to the search portion of the Discovery Methods module in four particular ways, one for each type of learning: (1) what parameters can/should be considered for modification (this is for parameter-modification learning), (2) which rules are candidates for specialization, (3) which rules should be modified in their conclusions (e.g., recommending different search strategies for rule-modification learning), and (4) when to attempt generalization.

Of course, there is a danger in providing domain knowledge to our system in that specifying such knowledge can rule out potentially valuable discovery. Frawley et al. (1992) point out the case in logistics planning where the search space is so large that it is impossible to find solutions without using constraints such as “trucks don’t drive on top of water (without bridges).” But adding this constraint eliminates potentially interesting solutions such as those in which trucks drive over frozen lakes in winter. So the key, they say, is to provide as general as possible constraints, while still maintaining enough specificity to provide useful discoveries.

There are four primary components in the Domain-Knowledge module; these may be modified or enhanced in the future. They are

- the performance measures component
- the low-confidence parameter list
- the link-of-influence submodule, and
- the classifier-methodology-to-search-space (CM/SS) component.

We now describe each of these components.

The performance-measures component contains the currently recommended measures for evaluating success in the KBSOS. At this point, we are utilizing the same performance measures as Crouch (1992), not because we have studied them and found them acceptable, but rather because we have focused our efforts elsewhere and have assumed them by default. (We believe this whole area to be a topic worthy of further study.) There are two Crouch performance levels, weak and strong, and both are defined in terms of what Crouch called “interesting” optimization sessions or cases. Two of Crouch’s three “interesting” cases are oriented toward the efficiency of the optimization, which Crouch measured according to the total number of runs used to find the optimal response. Those optimization sessions requiring relatively many runs are marked “Bad” or “B,” whereas those requiring relatively few runs are marked “Good” or “G.” The other Crouch “interesting” case is based upon effectiveness, which she measured by observing the variance of the surface and whether multiple optima exist. If there is high variance or if multiple optima exist, Crouch labels the case “Ugly” or “U.” We refer to Crouch’s three interesting cases as “BUG.”

As mentioned, Crouch then defined performance in terms of the BUG cases. Performance is judged as “strong” or “weak” according to the following two (Crouch) rules:

IF marked = G AND

marked < > U

THEN performance = strong;

IF marked = B

THEN performance = poor.

The performance measures “strong” and “weak” are used in the Discovery Methods module as will be explained shortly. With the modular structure of the domain-knowledge module, it is relatively easy to modify performance measures as desired.

Again, it is the purpose of the first of the four Domain-Knowledge module components, namely the performance-measures component, to provide the criteria whereby the success and failure of the KBSOS may be judged.

The second component in the Domain-Knowledge module is the Low-Confidence Parameter List. This list is simply a developer-supplied tabulation of the “important” parameters utilized in the rules. They are ranked according to the lack of confidence the developer has in their values, with least-confidence parameters at the top of the stack. When the Learner decides to attempt parameter modification, it will do so by popping the low-confidence-parameter-list stack, and considering the modification of the parameter at the top of that list using the parameter modification process flow outlined in Crouch (1992). Figure 2.10 shows where the KBSOS parameters that can be modified are located within the Classifier KBSOS.

The third aspect of the Domain-Knowledge module is the link-of-influence (LOI) submodule. The basic purpose of this component is to establish the link between any parameters to be modified and the effect on rules “downstream” in the knowledge base. For example, assume a given parameter in the “characterize” component in the classifier module in the KBSOS is presently set to a value of 0.5. If a change to 0.7 for this parameter is under consideration, then those cases (i.e., sessions) for which the parameter took on values between 0.5 and 0.7 must be re-examined. Now if the parameter being set at 0.7 in the “characterizer” caused a particular rule in the Strategy Selector to be fired and another rule in the Detailer subsequently to be fired, then the effect of the change to 0.7 must be considered to the extent that the downstream rules in the Selector and Detailer that would be fired instead of the initial set must be examined. For instance, the change from 0.5 to 0.7 might result in a whole new search strategy being chosen in the Selector.

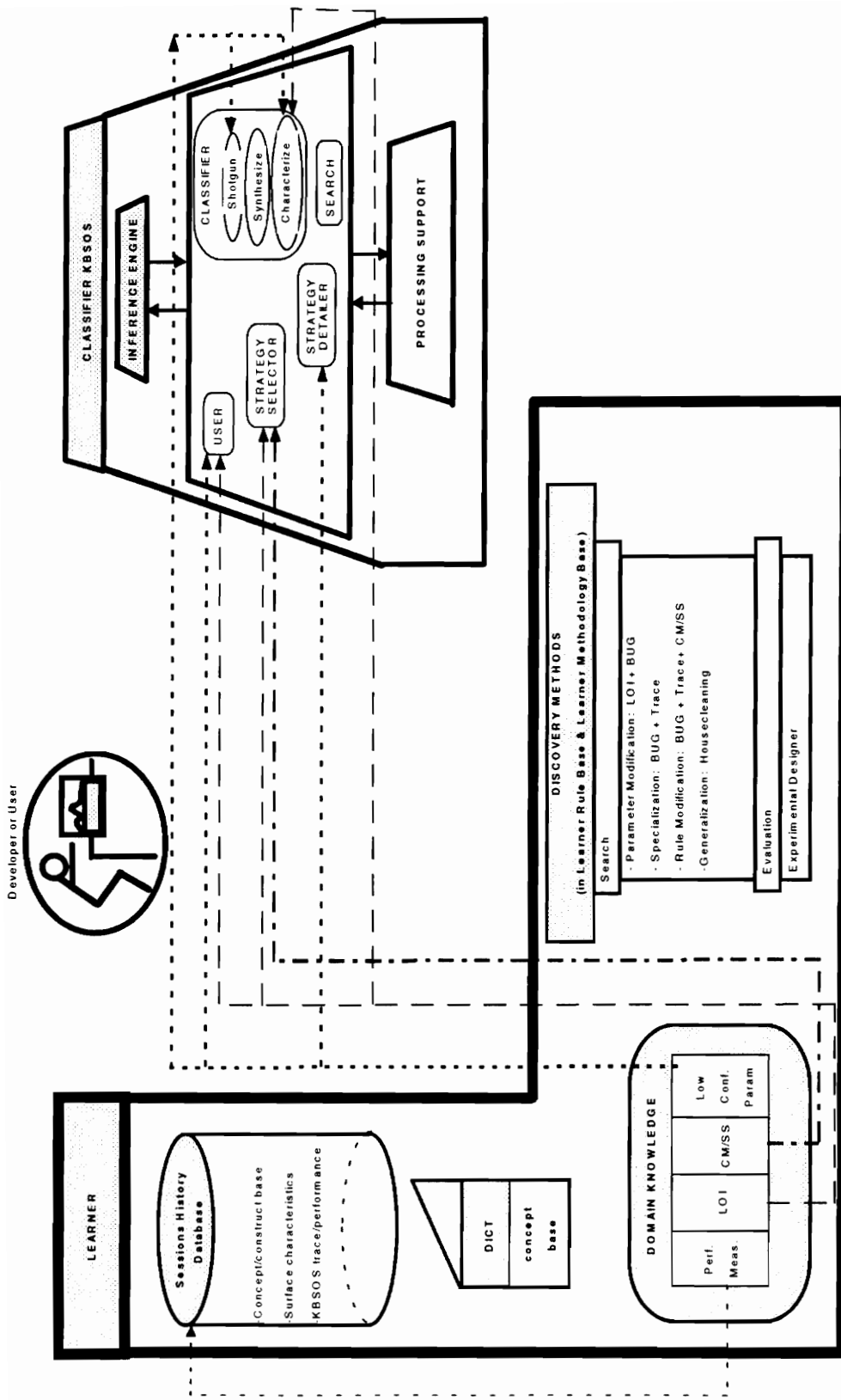
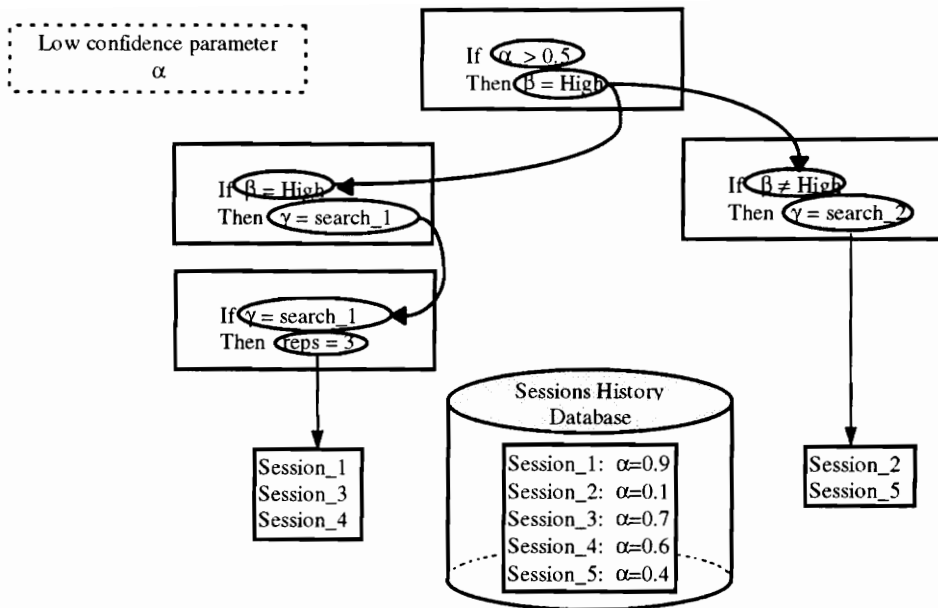


Figure 2.10. Details of the Domain Knowledge component of the Discovery Learner

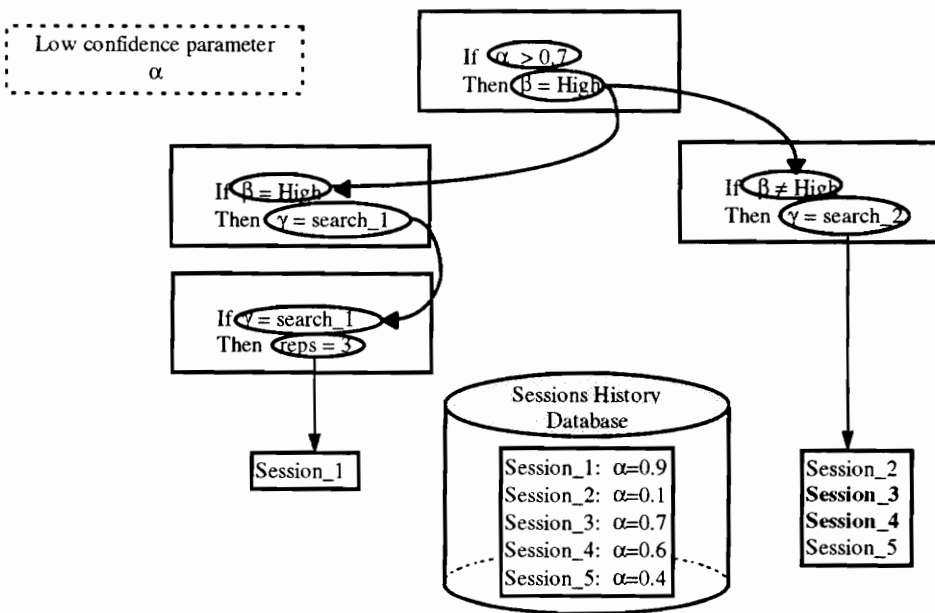
The determination of the downstream rules affected by a parameter shift is not difficult conceptually, as one merely needs to forward chain through the rules. Figure 2.11a shows how this works with a few rules and five sessions. The parameter α affects the parameter β which in turn affects the parameters γ . The parameter γ affects the number of replications but only for one search method. By forward chaining through the rules the parameters that are affected can be found. The threshold for parameter α is set at 0.5 and the rules that are affected by α are shown in figure 2.11a. The effect of changing the threshold from 0.5 to 0.7 is shown in figure 2.11b. Note that only two sessions (3 and 4) are impacted by the change. The particular modules in the Classifier KBSOS affected by the LOI submodule are also shown in figure 2.10 by the dashed lines leading from that submodule.

The final submodule currently present in the domain-knowledge portion of the Learner is the classifier-methodology-to-search-space (CM/SS) mapping. Recall that the Classifier KBSOS synthesizes simulation runs and then characterizes the resulting optimization surface according to six output measures. These output measures have been chosen particularly because they channel surfaces toward the search technique most appropriate for the type of surface.

The purpose of the CM/SS component as used in the Discovery Methods module is to suggest new search strategies for appropriate bad and ugly cases. Three current means of doing this in the CM/SS include what we call the “primitive method,” whereby Smith’s (1973) second and third search choices in his experiments are suggested; a taboo-region method, where those strategies deemed terrible in a particular region of classifier methodology/search space are listed as “to be avoided”; and a third method that calculates the Mahalanobis distance from the currently recommended strategy to the nearest centroid of the other strategies. As noted in figure 2.10, the CM/SS rules impact only the Strategy Selector module in the KBSOS.



2.11a. Initial Link-of-Influence



2.11b. Subsequent Link-of-Influence

Figure 2.11. Link-of-Influence

Discovery Methods module

The final Learner module to be discussed is the “work-horse” component, namely the Discovery Methods module. Recall that, as Frawley et al.’s (1992) paradigm suggests, discovery methods consist of search followed by evaluation. The search itself, they say, also has two parts: pattern identification and concept description. As mentioned, the former defines classes that maximize within-class similarity while minimizing among-class similarity. Concept description consists of deriving descriptions of the classes.

Our discovery method module also consists of search and evaluation, the latter of which we have labeled our “experimental designer” (in the sense of a “design-of-experiments” expert). The pattern identification phase of our search consists of the four tasks, parameter modification, specialization, rule modification, and generalization. The first three tasks are defined procedurally in Crouch (1992), and generalization is described in Greenwood et al. (1993). The procedures referenced are modified as explained in the example below. These four tasks are conducted instead of a more formal cluster analysis, although, in a sense, most of the four tasks pursue their goal through attempts at clustering BUG cases into clearer categories. The second portion of search, the concept description effort, utilizes rules as the representation scheme in which all new constructs will be expressed. This is both convenient, given that the four tasks are designed to operate on rules; and propitious for further discovery, since any rule in the KBSOS or Learner can, whether a new or an old construct, in principle, then be re-learned (i.e., modified, or even “unlearned,” etc.) by additional search using the four tasks.

Chapter Three: An Investigation of the Behavior of Simulation Response Surfaces

INTRODUCTION

Jacobson and Schruben (1989) point out that simulation optimization is in the class of stochastic optimization problems, where the objective function is a stochastic function of deterministic decision variables, thus making problems in this domain very difficult to solve. Much work has been done, as evidenced by the extensive literature on the subject, in refining the approaches to the simulation optimization problem. However, the literature does not provide a complete illustration of the complexity of simulation response surfaces and the difficulties they pose for optimizing a system. In this paper, we demonstrate how “messy” stochastic functions can be, even in the case of a simulation model that represents a fairly simple system. The stochastic nature of the system greatly confounds the search process. Our simulation experiment, which is conducted on a simple inventory model, and the ensuing discussion and graphical presentation of the results clearly illustrate the problems that one can encounter.

The behavior of the response surface is important to examine because it directly affects the choice of search technique as well as the degree of success obtained.

In order to guide the search process one needs descriptions of the behavior of the surface. We present a set of measures that are used to characterize the behavior of the surface and guide the search process. While these measures are not new, one contribution of this paper is to demonstrate the measures' use in the simulation optimization process in a unifying context; this is accomplished through a single comprehensive example.

The purpose of this paper is to explore the characteristics and properties of simulation surfaces with the focus on how the behavior of the surfaces affects the simulation optimization process. As part of this investigation, we identify and develop measures that characterize the behavior of the surfaces and discuss some implications if one insists on a statistically valid strategy at each step in the process. The behavior of the surfaces and the measures used to characterize that behavior are illustrated via the aforementioned inventory model.

This paper is organized as follows. The first section provides a brief background discussion of the simulation optimization problem. It is followed by a description of the model and the experiments that are performed in order to illustrate the behavior of the simulation response surfaces. The third section defines a series of measures, both point estimates and regional measures, that are used to characterize a simulation model's behavior. The findings of this research are then discussed and presented graphically to provide insight into how the response surfaces of a simple simulation model behave under varying degrees of variability and design conditions. The results illustrate how these conditions affect the choice of a search technique/methodology in the simulation optimization process. The final section discusses the implications that the findings pose for future research in simulation optimization.

DEFINITION OF THE SIMULATION OPTIMIZATION PROBLEM

Simulation is commonly recognized as one of the most widely applied computer modeling techniques in use today. Its popularity is evidenced by the large number of applications documented in the literature and the extensive breadth of problem domains to which it has been applied. With the advent of rapidly advancing computer technology, the widespread use of simulation is expected to accelerate. The value of simulation is that it permits the study of systems which cannot feasibly be constructed or experimented upon in the “real world,” and which are too complex to be analytically modeled. Simulation is very useful in predicting the output of a system or its response to a given set of input conditions. However, it does not in and of itself indicate the input conditions required to achieve a desired response. Simulation is an evaluative methodology and not an optimization technique.

In many cases the strategic objective of a study is to find the best solution for the system under investigation, i.e., optimize the system’s performance. When the search for the optimal solution involves the use of data obtained from a simulation model, the analysis involves the process of *simulation optimization*. The optimization process is complicated by the presence of random error, often the result of the combined random effect of uncontrollable conditions.

Note that we refer above to simulation optimization as a *process* and not as a technique, methodology, or algorithm. In fact, the process of simulation optimization typically utilizes a wide range of mathematical and statistical tools. There is no single or standard approach to optimizing a system where the data for the analysis is based on experiments conducted with a simulation model. Some approaches focus on a single simulation run (e.g., frequency-domain analysis, perturbation analysis). Others focus on a search process that involves multiple simulation runs. Within this approach, which is the most common, there are many philosophies on how the search should be conducted. For example many approaches utilize the

data to fit metamodels (e.g., response surface methodology, neural networks, nonparametric regression); others are free of underlying model assumptions (e.g., random search, Box's complex search, genetic algorithms). Yet another approach (Crouch, Greenwood, Rees, 1995) (Greenwood, Rees, Crouch, 1993) proposes a multi-strategy process that utilizes the "best" methodology based on current experimental and synthesized knowledge of the search environment. This brief discussion of the approaches to simulation optimization is meant to illustrate the diverse and varied literature that exists to solve this difficult problem. It is beyond the scope of this paper to review all of these approaches to simulation optimization. Therefore, the interested reader should refer to overview or literature review articles and introductory texts on the subject, e.g., (Azadivar, 1992) (Barton, 1992) (Jacobson, Schruben, 1989) (Meketon, 1987) (Myers, 1971) (Safizadeh, 1990).

In general, the simulation optimization problem can be expressed as:

$$\text{Optimize:} \quad E[\mathbf{Y}] = E[f(\mathbf{X} | \mathbf{Z})] \quad (1)$$

$$\text{Subject to:} \quad \mathbf{h}(\mathbf{X}) \leq \mathbf{0} \quad (2)$$

where the responses that are to be optimized, $\mathbf{Y} = (Y_1, Y_2, \dots, Y_m)$, are functions of controllable factors, $\mathbf{X} = (X_1, X_2, \dots, X_n)$, uncontrollable conditions, \mathbf{Z} , and random error, ϵ ; i.e., $\mathbf{Y} = E[\mathbf{Y}] + \epsilon = f(\mathbf{X} | \mathbf{Z}) = E[f(\mathbf{X} | \mathbf{Z})] + \epsilon$. Each response Y_j is a random variable and takes on a set of values for the same setting of the controllable factors; i.e., there is some distribution of Y_j values for each combined level of the controllable factors. To model this behavior each response is oftentimes considered equal to the sum of a constant and a noise term that represents the random error, where the constant is the expected value of the response, $E[Y_j]$, for a specific combination of factor settings. Therefore, due to the presence of random error, the optimization process typically focuses on the expected value of the responses. But, while the goal is to optimize $E[Y_j]$, only Y_j is observable. Also, each objective regarding Y_j involves either the absolute maximization (or minimization) of Y_j or the achievement of Y_j to exceed some goal by a specified tolerance. The simulation optimization problem is constrained, at least by the bounds of the

region to be explored. As shown in (2), $\mathbf{h}(\mathbf{X})$ is a vector of deterministic constraints typically of the form: $L \leq \mathbf{X} \leq U$ or $L \leq f(\mathbf{X}) \leq U$, where L and U are the lower and upper bounds of the search region, respectively. Typically the regional boundaries change as the search process progresses. For example, as more information becomes known about the search environment and characteristics of the surface, the search region narrows so as to include only the most promising sector(s). The domain of the region may be either continuous, discrete, or mixed.

DEFINITION OF THE EXEMPLARY MODEL AND EXPERIMENTS

The concepts presented in this paper are demonstrated through experimentation with a simple inventory model that permits backorders. Experimentation with this model illustrates the effect of changes in the model's parameters on its simulated response surface. This section defines both the model and the experimental conditions that are considered.

Simulated inventory system

The model is analogous to the continuous-review EOQ model, except that it is stochastic. The inventory model, illustrated in Figure 3.1, contains two decision variables or controllable factors -- order quantity (Q) and re-order point (R). They are varied during the search process in order to find the combination of

Q and R that yield the lowest total cost (TC). Total cost is composed of three components: ordering cost, carrying or holding cost, and shortage or backorder cost.

The model contains two uncontrollable conditions, demand and lead time -- these are random variables, and not constants, as assumed in the basic EOQ model. Demand (D) is a random variable and causes the inventory level to decrease at a non-constant rate, as illustrated in Figure 3.1. Lead time (L), the time between order placement and order receipt, is also a random variable. The effect of the stochastic lead time is that the inventory level does not always return to the same maximum value when an order of size Q is received, as illustrated in Figure 3.1.

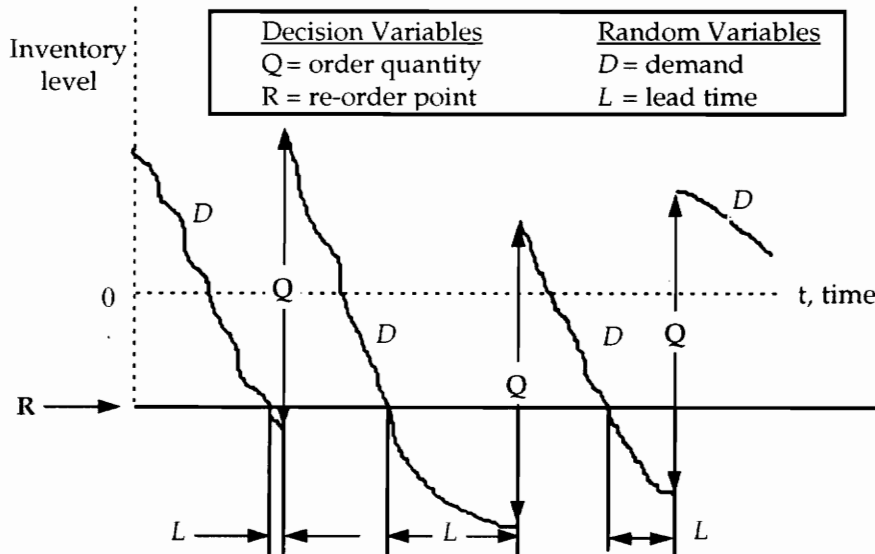


Figure 3.1. Simple inventory model that permits backorders and exhibits both stochastic demand and lead time. Note: Q and R, the decision variables, are fixed in any given simulation run.

The problem may be stated as:

$$\text{Optimize:} \quad \text{Minimize } \{ E[TC] = E[f(Q, R | D, L)] \} \quad (3)$$

$$\text{where } \frac{1}{D} \sim \text{Gamma}(\alpha_D, \beta_D) \text{ and } L \sim N(\mu_L, \sigma_L^2),$$

α and β are shape and scale parameters, respectively,

μ and σ^2 are the mean and variance of a normal distribution,

$\frac{1}{D}$ is the time between demands on the system, and

L is the time between order placement and order arrival.

Subject to: $0 < Q \leq 400$ (4)

$-400 \leq R \leq 0$ (5)

$|R| \leq Q$. (6)

The first constraint, as shown in (4), defines the initial estimate of the domain of the order quantity, Q ; i.e., it is assumed the “optimum” order quantity will be less than or equal to 400 units. This is based on the decision maker’s understanding of the problem and values of such cost parameters as the cost to place an order, cost of one unit to be in inventory for one year, etc. The second constraint, in (5), limits the value of the second decision variable R , re-order point. In this example, an order will be placed when the inventory level reaches zero, when the number of back orders reaches 400, or somewhere in between. The final constraint (6) ensures that a policy where the system is always in a backorder situation is avoided. This would occur if Q was not set large enough to meet all backorders in an order cycle, on the average. Note that this constraint restricts the feasible region to be triangular.

The simulation model of the inventory system operates as follows. The times between single-item demands on the system, $\frac{1}{D}$, are randomly generated based on samples from a Gamma(α , β) distribution.

If the request for demand cannot be met from on-hand inventory, it is considered backordered. This unsatisfied demand is filled immediately upon replenishment of the inventory -- when an inventory “order” arrives. An order, of size Q , is placed when a demand arrival causes the inventory position to reach (or go below) the reorder point, R . The order will arrive L days after the order is placed, where each

L is randomly generated based on a sample from the truncated Normal distribution (i.e., L is not permitted to go negative).

The process that is followed in the simulation optimization process is illustrated in Figure 3.2. The operation of the system, as represented by the simulation model, is run for a specified period of time. The performance of the system is based on total cost, an output of the simulation model and the response that is to be minimized. Total cost is based on the specified values of the decision variables or controllable factors -- order quantity and re-order point, Q and R, respectively -- and random demand and lead time values that occurred during the simulated operation of the system. Every possible combination of Q and R, i.e., every point in (Q, R) space, represents a possible simulation run. In order to improve upon the expected total cost of the system, one changes the values of the decision variables and simulates the operation of the system again at another (Q, R) location. Decisions on how to change the value of the decision variables in order to get an improved solution occur in the "optimizer" box in Figure 3.2. The optimizer may involve a simple random strategy or a more complex but rational approach such as response surface methodology. Multiple simulation runs, replications, may be made at a single (Q, R) point in order to obtain a better estimate of the response, total cost, and obtain an estimate of the variability of the response.

In order for the model to provide results that are comparable across a variety of scenarios, the simulation run duration must span complete order cycles and not a fixed period of time. Since every (Q, R) combination results in a different order cycle, failure to account for "end effects" (stopping the simulation at different points in the order cycle) would bias the estimate of total cost. Therefore, the actual length of a simulation run is the intended run length (e.g., one-half year, four years) plus whatever time is necessary to complete the last order cycle. In addition, each simulation run includes a "warm up" period of operation before statistics are collected. In order to illustrate these run-time controls, consider the simulation of a (Q, R) inventory system that is to be evaluated based on four years of operation following a

one-year “warm up” (assume 250 days per year). The simulation begins at a point immediately after an order arrives (time zero) and continues until the first cycle beyond 250 days is completed, say 251.2 days. The statistical arrays are cleared at this point and the simulation is run for at least another four years, at least until time 1251.2. The simulation terminates at the end of the first order cycle beyond this point, say another 2.3 days. Therefore, while the total simulation time is 1254.5 days, the performance measures would be based on a simulated time of 1002.3 days -- total simulated time less warm-up time.

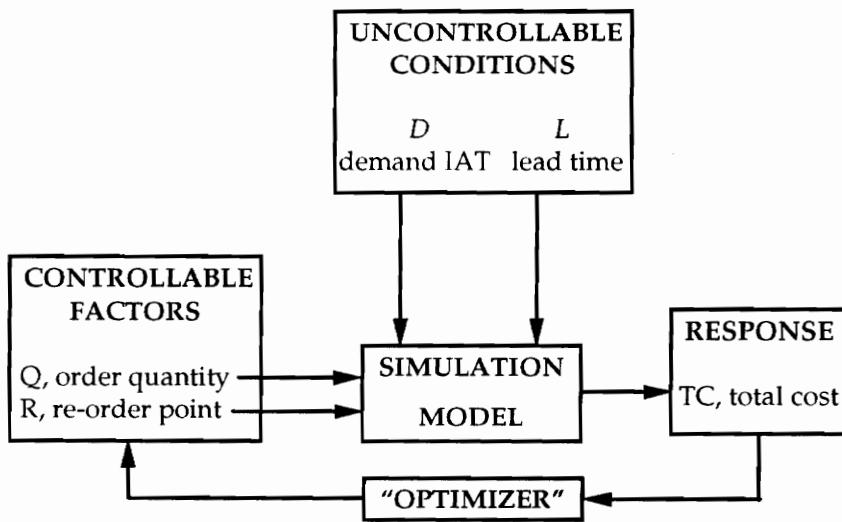


Figure 3.2. Process for optimizing the simulated inventory system

The single response or performance measure ($Y=Y_1$) considered in this example is TC, “total average inventory cost per day” over the simulated time period. As shown in (7), the cost measure is composed of three components -- ordering cost, carrying cost, and shortage cost. Ordering cost is the product of the number of orders placed during the simulation (O) and the cost to place an order (C_o). Carrying costs is the product of average inventory during the simulation (\bar{I}) and the cost of carrying an item in inventory per period (C_c). Shortage cost is the product of the average number of units short or backordered during the simulation (\bar{S}) and the cost of being short one period (C_s). For the model discussed in this paper, C_o

= \$50, $C_c = \$10/\text{item}/\text{year}$, and $C_s = \$5/\text{item}/\text{year}$. The length of the simulation T is the intended run length, after warm up, plus whatever time is necessary to complete an order cycle (with regard to the example given above, $T = 1002.3$ days.)

$$TC = \frac{O * C_o + \bar{I} * C_c + \bar{S} * C_s}{T} \quad (7)$$

For comparison purposes later in the analysis, the analytic solution to this problem, assuming deterministic demand and lead time, is $Q = 194$, $R = -129$, and $TC = \$2.58$ per day.

Experimental conditions

As mentioned above, the random variable demand is assumed to follow a Gamma(α , β) distribution, where the mean demand has a value of $\alpha\beta$ and a variance of $\alpha\beta^2$. The random variable lead time follows a normal distribution with mean μ and variance σ^2 , with no lead-time permitted to be negative. In all the cases considered in the paper, the mean of the random variable remains constant but the variance is changed in order to illustrate the effect of variability on the search process. As shown in Table 3.1, experiments are run where the mean time between requests for demand is 0.2 days (conversely, mean demand is 5 units per day). Variability in demand is considered “low” and “high” -- the coefficient of variation (CV) of time between requests for demand is 1.00 and 4.5. Likewise, mean lead time is 6 days and is either considered to exhibit “no” or “moderate” variability, corresponding to a coefficient of variability of 0.00 or 0.33 (Normally distributed with a mean of 6 days and a standard deviation of 2 days). The no variability in demand and no variability in lead time case -- not shown in the table -- was used to validate the simulation model by comparing its results to the analytic solution. Results of this case are not reported here.

Table 3.1. Definition of experimental cases

			DESIGN CONDITIONS		
			BEST (# REPLICATIONS=10; RUN LENGTH = 4 YRS)	WORST (# REPLICATIONS=3; RUN LENGTH = 0.5 YRS)	
VARIABILITY OF DEMAND	LOW (CV=1.0)	VARIABILITY OF LEAD TIME	NONE (CV=0.0)	(LNB)	(LNW)
		VARIABILITY OF LEAD TIME	MODERATE (CV=0.33)	—	—
	HIGH (CV=4.5)	VARIABILITY OF LEAD TIME	NONE (CV=0.0)	(HNB)	—
		VARIABILITY OF LEAD TIME	MODERATE (CV=0.33)	—	(HMW)

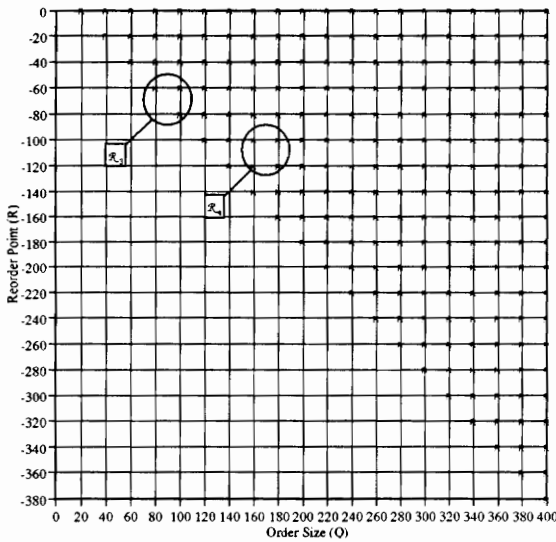
The behavior of the model's simulated response surface is also sensitive to two design variables -- the number of replications and the length of time the model is simulated. Each of these variables is considered at two levels. In this example, three replications is considered "small" and ten replications is considered "large." Also, running the simulation model for a six-month period is considered "short" and running the model for four years is considered "long." On the one hand, one can think of the case where the model is run for six-months and replicated three times as the "worst" case, i.e., the one that would produce the "messiest" surface. On the other hand, one can consider the case where the simulation model is run for four years and replicated ten times as the "best" case.

In order to illustrate the behavior of simulation response surfaces, it is not necessary to run all combinations of the variables under consideration. This paper, in a later section, reports the results for four experimentation conditions or four cases -- selected combinations of inventory model variabilities and design conditions. The first two cases, referred to as LNB and LNW, both examine Low demand

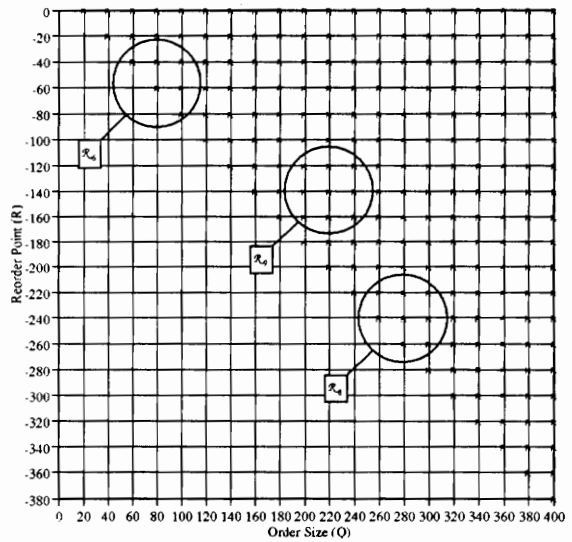
variability and No lead-time variability; the first case explores the Best design conditions we consider, whereas the second explores the Worst design conditions. The third case, HNB, considers high demand variability with no lead-time fluctuation and the best design-condition set. The fourth case, HMW, explores high demand variability, moderate lead-time variability, and the worst design conditions. These four cases are summarized in Table 1. Note that by comparing LNB with LNW some feeling of the effect of design conditions can be seen; comparing LNB with HNB shows the effect of demand variability, whereas contrasting LNW and HMW illustrates the effect of both demand and lead-time variabilities simultaneously; and comparing LNB with HMW shows the result of “better” versus “worse” conditions for all three factors.

Data Collection Scheme

Figure 3.3 shows the triangular decision-variable or search space for the problem. The space is defined by the constraints of the inventory model, defined in (3) through (5) above.



3.3a. Δ -by- Δ regions



3.3b. 2Δ -by- 2Δ regions

Figure 3.3. Inventory model’s decision space

Point-estimate statistics of the simulation-response surface are collected in a grid pattern over the decision variable space $\mathbf{X} = (x_1, x_2)$. The gridpoints are spaced 20 units apart in each direction, i.e., $\Delta_1 = \Delta_2 = \Delta = 20$. A plot of the grid spacing is shown in Figure 3.3.

In order to develop statistical estimates over a region, the gridpoints are combined to form a region of analysis. For a grid spacing Δ , each gridpoint forms a corner of a region (no region will be larger than Δ by Δ). Most of the regions will be Δ by Δ square, with a gridpoint at each corner. However, because the model's feasible region is triangular, some regions formed (in particular, those along the line $x_1 = x_2$) will themselves be triangular, and will be defined by only three gridpoints. These two situations are illustrated in Figure 3.3a. For example, regions with three gridpoints, designated here as \mathcal{R}_3 are all of the general shape

$$\mathcal{R}_3 = \begin{pmatrix} * & * \\ & * \end{pmatrix}, \text{ where the } * \text{ denotes the coordinates } (\zeta_1, \zeta_2) \text{ of a gridpoint. Specifically,}$$

$$\mathcal{R}_3 = \begin{pmatrix} (\zeta_{1i} + \Delta, \zeta_{2i}) & (\zeta_{1i} + 2\Delta, \zeta_{2i}) \\ & (\zeta_{1i} + 2\Delta, \zeta_{2i} - \Delta) \end{pmatrix}, \text{ where } 0 \leq \zeta_{1i} < 400 - \Delta, \text{ and } \zeta_{2i} = -\zeta_{1i}.$$

In this research, we also consider the effect of larger regions, ones that are 2Δ by 2Δ . Most of these regions will be squares that contain nine gridpoints, denoted by \mathcal{R}_9 . However, due to the shape of the decision space, six- (\mathcal{R}_6) and eight-point (\mathcal{R}_8) regions are also possible. Examples are shown in Figure 3.3b.

STATISTICAL MEASURES OF THE BEHAVIOR OF SIMULATION SURFACES

In order to assess the behavior of a simulation surface, measures need to be defined that capture the salient behavior of the surface. One measure is the value of the response itself, total cost, at each (Q, R) point simulated. If that point is replicated -- i.e., multiple independent simulation runs are made at point (Q, R) -- then an obvious measure is the mean response at that point. Other point measures considered in this paper, as defined below, include the following statistics that measure dispersion at the point (Q, R): standard deviation, coefficient of variation, and signal-to-noise ratio. Another point measure considers the "relative activity" or acceleration of the surface. In addition to point measures, one is also interested in region statistics, often used to check the validity of using a particular parametric test or utilizing a procedure that assumes the surface exhibits a particular characteristic over the search region, e.g., homogeneity of variance.

By considering the various measures of the behavior of the simulation response surface, the intent is to characterize the surface in such a way that it is useful in determining the appropriate search strategy to employ. For example, if measurements of the region of interest indicate significantly different variances, then it may not be prudent to use an optimization strategy such as Response Surface Methodology (RSM) (Myers, 1971) that assumes homogeneity of variance throughout the region.

POINT-ESTIMATE MEASURES

Each simulation run provides information on the performance or response of the system to a set of input conditions. The output of a simulation provides an estimate of the response at a single point in the decision space $\mathbf{X} \subset \mathbb{R}^n$. In this example, $X_i \subset \mathbf{X} = (X_1, X_2) = (Q, R) \subset \mathbb{R}^2$ represents one combination

of order quantity and re-order point and the location of one simulation run. For the measures defined below, the subscript i indicates the measure was estimated at the point X_i in the decision space. Since the system being simulated is stochastic, multiple simulation runs at the same point, i.e., replications, will result in different responses at the same point. A realization j of the single response total cost (TC) is denoted as y_j . The number of replications at each point, r_i , is assumed to be constant; i.e., $r_i = r \forall i$.

Location Measures

Only one location measure is explored: the arithmetic mean of the replicated response(s) at a given point

$$\bar{y}_i = \frac{1}{r} \sum_{j=1}^r y_{ij} \quad (8)$$

Dispersion Measures

Three dispersion measures are considered at points i in X : (1) the response standard deviation s_i , (2) the response coefficient of variation CV_i , and (3) the signal-to-noise ratio of the response $(S/N)_i$ (see, for example, (Barton, 1992), pages 289-299). The measures are defined as:

$$s_i = \frac{1}{(r-1)} \sum_{j=1}^r (y_{ij} - \bar{y}_i)^2, r > 1 \quad (9)$$

$$CV_i = \frac{s_i}{\bar{y}_i}, \bar{y}_i \neq 0, \text{ and} \quad (10)$$

$$\left(\frac{S}{N}\right)_i = 10\log_{10} \left[\frac{1 - \sum_{j=1}^r y_{ij}^2}{s_i^2} \right], r > 1, s_i > 0. \quad (11)$$

Relative Activity

There are two components of curvature in a response surface: actual surface undulations and variance at each point. Actual undulations are of interest because they indicate where optima are. The variance at each point can mask or heighten the perceived activity. One goal of functional synthesis is to find true areas of curvature in a function. Good and Gaskin (1971) suggest using the second derivative to locate areas of a function that have more relative activity. Müller (1984) suggests spending more runs at points that have a lot of relative activity, i.e., where the second derivative is large. Since in simulation optimization, response surface values are known only at discrete points where simulation runs are made, an approximation of the second derivative must be made. A numerical analysis estimate (Conte, de Boor, 1980) is used:

$$f''(a) = \frac{f(a-h) - 2f(a) + f(a+h)}{h^2}. \quad (12)$$

Since h does not approach zero in our application, the approximation to the second partial derivative is very coarse. Therefore, we refer to f'' as “bumpiness” and not the second derivative.

Although functional synthesis is used to get a good estimate for a function across the *entire* function, what is relevant to simulation optimization is (1) where we need to spend more points to accurately detect activity, and (2) is the point a local optimum in the “proper direction” (e.g., is it a local maximum in a

maximization problem). Obviously, we do not care about local minima in a maximization problem. We are interested in spending more points where there appears to be a local optimum in the desired direction.

Computationally, a local optimum can be found by combining the second derivative with the first derivative. If the first derivative at a point is equal to zero, then a local optimum exists there. The second derivative can then be used to determine if the point is a maximum or a minimum. If the first derivative is not equal to zero at a point, then first derivatives are checked at adjacent points. If the first derivatives show that zero is crossed from one side of the point to the other (a change in sign), then a local optimum has been found.

In the plots presented below, four different types of relative-activity graphs are shown. The first is the bumpiness in the Q-direction, with R held constant; the second is the bumpiness in the R-direction, with Q held constant. The third shows the bumpiness only where the first derivative is zero or crosses zero in the Q-direction, and the fourth plots the same combined measure in the R-direction.

REGIONAL MEASURES

In searching for the “optimal” solution, and deciding how to conduct that search, it is helpful to obtain information in the decision space. Point estimates are combined to provide estimates of how the response surface behaves across some portion of the decision space. A region is defined as adjacent points in the decision space where a group of simulation runs are made, $\mathcal{R}_k \subset \bigcup_{\forall i} X_i$. Regions for this example were defined earlier and include: $\mathcal{R}_3, \mathcal{R}_4, \mathcal{R}_6, \mathcal{R}_8, \mathcal{R}_9$.

Of the four regional estimates presented below, the first two are used to test normality and homogeneity of variance assumptions. Most parametric statistical techniques, e.g., regression and RSM, are based on

these assumptions. The second two measures provide an assessment of how well a plane or hyperplane would fit the data contained in the region.

Test for Normality

The question addressed with this measure is whether the residuals obtained after fitting a hyperplane over a region (i.e., \mathcal{R}_3 , \mathcal{R}_4 , \mathcal{R}_6 , \mathcal{R}_8 , \mathcal{R}_9) are normal. This is an important question because normality of the residuals is an implicit assumption commonly used in simulation optimization F-tests such as “significance of regression” and “lack of fit” (see below).

The Shapiro-Wilk test for normality is used here because it is a powerful omnibus test, i.e., it is a test that will test the normal distribution against any alternative distribution (D’Agostino, Stephens, 1986). The hypothesis tested is:

Hypotheses:

H_0 : The distribution of all the residuals in the region \mathcal{R} , obtained after fitting a hyperplane through all responses observed in the region, follows a normal distribution.

H_1 : The distribution of residuals does not follow a normal distribution.

Test Statistic:

$$W = \frac{(\sum a\epsilon)^2}{\sum (\epsilon - \bar{\epsilon})^2},$$

where “a” is a tabulated constant (see, e.g., (D’Agostino, Stephens, 1986) pp. 209-211), and ϵ represents the residuals (sorted in ascending order) in the region.

Decision Rule:

Reject H_0 if $W < W^*$, where W^* is tabulated and may be found, e.g., in (D’Agostino, Stephens, 1986), pp. 212-213. Fail to reject H_0 otherwise.

Test for Homogeneous Variances

This test investigates whether variances are homogeneous across a region \mathcal{R} of interest. This issue is important for the same reason that the normality test is, namely that homogeneity of variance is an assumption of F-tests.

The procedure initialized to examine this issue is as follows. First, a region \mathcal{R} (either \mathcal{R}_3 , \mathcal{R}_4 , \mathcal{R}_6 , \mathcal{R}_8 , or \mathcal{R}_9) is specified. Next the variance is calculated (using the replications) at each design point in the region. For example, in a region with six points (\mathcal{R}_6) with ten replications at each point, six separate variance calculations are made, each involving ten responses. Since the proper homogeneity-of-variance test depends on whether region residuals are normal, the Shapiro-Wilk normality test described above is performed over the region. If the residuals can be safely assumed to come from a normal distribution, then Bartlett's test, with Box's Transformation is used to examine homogeneity of variances in the region; if they cannot be assumed to be normal, Levene's Test (using the median rather than the mean) should be used.

Each of these tests is now described.

Bartlett's Test with Box's Transformation. (Neter, Wasserman, Kutner (1985), pp. 618-622.) Since Bartlett's test is sensitive to departures from normality it will only be used if the Shapiro-Wilk test for normality is not rejected.

Hypotheses:

$$H_0: \sigma_1^2 = \sigma_2^2 = \dots = \sigma_g^2$$

$$H_1: \text{not all } \sigma_i^2 \text{ are equal,}$$

where g is the number of gridpoints in the region,

r is the number of replications, and

$n = gr =$ total number of runs in the region.

Test Statistic:

$$B = \frac{1}{C} \left[(df_T) \log_e \text{MSE} - \sum_{i=1}^g (df_i) \log_e s_i^2 \right]$$

$$C = 1 + \frac{1}{3(g-1)} \left[\left(\sum_{i=1}^g \frac{1}{df_i} \right) - \frac{1}{df_T} \right]$$

$$\text{MSE} = \frac{1}{df_T} \sum_{i=1}^g df_i s_i^2$$

$$s_i^2 = \frac{1}{r-1} \sum_{j=1}^r (y_{ij} - \bar{y}_i)^2$$

$$df_i = (r-1)$$

$$df_T = (r-1)g$$

In the above B is approximately distributed as χ^2 with $(g-1)$ degrees of freedom. Box's transformation is used to accommodate cases where the number of replications is less than four (i.e., $r < 4$). The following approximation can be "used when some of the degrees of freedom are small and [it] ... also is appropriate for large degrees of freedom" (Neter, Wasserman, Kutner, 1985, p. 620).

$$B' = \frac{f_2 BC}{f_1 (A - BC)}$$

where:

$$f_1 = g - 1$$

$$f_2 = \frac{g + 1}{(C - 1)^2}$$

$$A = \frac{f_2}{2 - C + \frac{2}{f_2}}$$

and B and C are as defined as above.

B' is approximately distributed as $F(f_1, f_2)$.

Decision Rule:

Reject H_0 if $B' > F(1 - \alpha; f_1, f_2)$. This means that the data come from populations that do not have common variance. Rejection means that there is at least one gridpoint whose variance is different from the others in the region (heteroskedasticity). Failure to reject means that there is not enough evidence to say that the variances are not different (homoskedasticity).

Levene's Test (with median). (Glaser, 1983) Conover et al. (1981) list this test as one of three that are superior in terms of robustness and power.

Hypotheses:

$$H_0: \sigma_1^2 = \sigma_2^2 = \dots = \sigma_g^2$$

$$H_1: \text{not all } \sigma_i^2 \text{ are equal,}$$

where g, r, and n (see below) are defined as above.

Test Statistic:

$$W = \frac{(n - g) \sum_{i=1}^g r (\bar{Z}_i - \bar{Z}_{..})^2}{(g - 1) \sum_{i=1}^g \sum_{j=1}^r (Z_{ij} - \bar{Z}_i)^2},$$

where $Z_{ij} = |y_{ij} - \tilde{y}_i|$ and where \tilde{y}_i is the median.

Decision Rule:

Reject H_0 if $W > F(g - 1, n - g)$. This means that the data come from populations that do not have common variance. Rejection means that there is at least one gridpoint variance that is different from the others (heteroskedasticity). Failure to reject means that there is not enough evidence to say that the variances are not different (homoskedasticity).

The next two regional measures are based on the following first-order regression model being fit over the region of interest, $y = G\beta + \varepsilon$, and the associated ANOVA:

SOURCE	DF	SUMS OF SQUARES	MEAN SQUARE	F-RATIO
Regression (R)	p-1	$\hat{\beta}'G'y - \frac{1}{n}y'J_n y$	$MS_R = \frac{1}{(p-1)} \left(\hat{\beta}'G'y - \frac{1}{n}y'J_n y \right)$	$F_R = \frac{MS_R}{MS_E}$
Error (E)	n-p	$y'y - \hat{\beta}'G'y$	$MS_E = \frac{1}{(n-p)} (y'y - \hat{\beta}'G'y)$	
Lack of Fit (LOF)	g-p	$y'y - \hat{\beta}'G'y - \frac{1}{g}y'\mathcal{S}_g$	$MS_{LOF} = \frac{1}{(g-p)} \left(y'y - \hat{\beta}'G'y - \frac{1}{g}y'\mathcal{S}_g \right)$	$F_{LOF} = \frac{MS_{LOF}}{MS_{PE}}$
Pure Error (PE)	n-g	$\frac{1}{g}y'\mathcal{S}_g$	$MS_{PE} = \frac{1}{g(n-g)} y'\mathcal{S}_g$	
TOTAL	n-1	$y'y - \frac{1}{n}y'J_n y$		

where

p = the number of parameters to be estimated in the regression model,

g = the number of distinct sets of levels for the X variables,

n = the number of observations (including) replications,

y = a vector of n observations of responses,

G = the gridpoint region design matrix.

Design matrices G_g are defined for each type of region defined above ($\mathcal{R}_3, \mathcal{R}_4, \mathcal{R}_6, \mathcal{R}_8, \text{ or } \mathcal{R}_9$) and are based on the number of gridpoints ($g = 3, 4, 6, 8, 9$) in that region; for example, for $g = 3$ (which implies $0 < \zeta_{1i} < 400 - \Delta$ and $\zeta_{2i} = -\zeta_{1i}$) with four replicates,

$$G_3 = \begin{pmatrix} 1 & \zeta_{1i} & \zeta_{2i} \\ 1 & \zeta_{1i} + 2\Delta & \zeta_{2i} \\ 1 & \zeta_{1i} + 2\Delta & \zeta_{2i} - \Delta \\ \dots & \dots & \dots \\ 1 & \zeta_{1i} & \zeta_{2i} \\ 1 & \zeta_{1i} + 2\Delta & \zeta_{2i} \\ 1 & \zeta_{1i} + 2\Delta & \zeta_{2i} - \Delta \\ \dots & \dots & \dots \\ 1 & \zeta_{1i} & \zeta_{2i} \\ 1 & \zeta_{1i} + 2\Delta & \zeta_{2i} \\ 1 & \zeta_{1i} + 2\Delta & \zeta_{2i} - \Delta \\ \dots & \dots & \dots \\ 1 & \zeta_{1i} & \zeta_{2i} \\ 1 & \zeta_{1i} + 2\Delta & \zeta_{2i} \\ 1 & \zeta_{1i} + 2\Delta & \zeta_{2i} - \Delta \end{pmatrix},$$

where

$\hat{\beta}$ = the vector of ordinary least-mean squares estimates of regression model parameters

given by $\hat{\beta} = (G'G)^{-1}(G'y)$,

J_1 = the $t \times t$ matrix of all 1s, and

$$S_g = \begin{pmatrix} J_g & 0 \\ 0 & J_g \end{pmatrix}.$$

Similar definitions may be made for $G_4, G_6, G_8, \text{ and } G_9$.

F-test for Significance of Regression

This test determines whether a relationship exists between the dependent variable and any of the independent variables. The test assumes normality of the error ϵ in the regression model and homogeneity of variance across the region.

Hypotheses:

$$H_0: \beta_i = 0, \forall i \ni i = 1, \dots, p - 1$$

$$H_1: \beta_i \neq 0, \text{ for at least one } i, i = 1, \dots, p-1.$$

Test Statistic:

$$F_R = \frac{MS_R}{MS_E},$$

Decision Rule:

If $F_R > F_{1-\alpha, p-1, n-p}$, then the null hypothesis is rejected at the level α , and it is concluded that there is a significant relationship between X and y in that region. Failure to reject H_0 implies that the hyperplane fit with the β_i is flat (or horizontal), i.e., no significant relationship appears to exist between the dependent variable and any independent variable.

F-test for Lack of Fit

If there is a relationship between the dependent variable and at least one of the independent variables, it becomes important to know whether the postulated linear regression model *adequately* fits the data. As with the previous test, normality and homogeneity of variance are assumed.

Hypotheses:

$$H_0: E(y) = \beta_0 + \beta_1 X_1 + \dots + \beta_{p-1}$$

$$H_1: E(y) \neq \beta_0 + \beta_1 X_1 + \dots + \beta_{p-1}$$

Test Statistic:

$$F_{LOF} = \frac{MS_{LOF}}{MS_{PE}}$$

Decision Rule:

If $F_{LOF} > F_{1-\alpha, g-p, n-g}$, then the null hypothesis is rejected at the level α . That is, rejecting the null hypothesis implies that a plane cannot be accurately fit through the (X, y) points over the region.

EXPLORATION

This section discusses the results, outcomes, and insights gained from this research. The discussions are based on graphs of the statistical measures over the decision-variables or search space. The study is based on the simple inventory system presented above that permits backorders, where each measure is compared across a variety of scenarios. These scenarios include both changes in the simulation design (run length and number of replications) and changes in the inherent variability in the system (due to variation in the demand and lead-time processes).

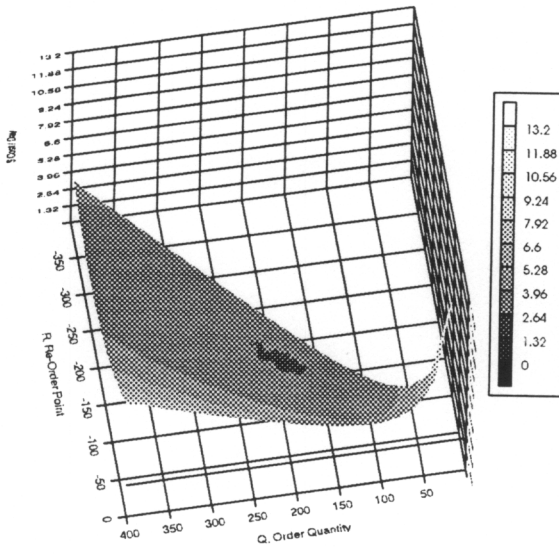
POINT ESTIMATES

Mean

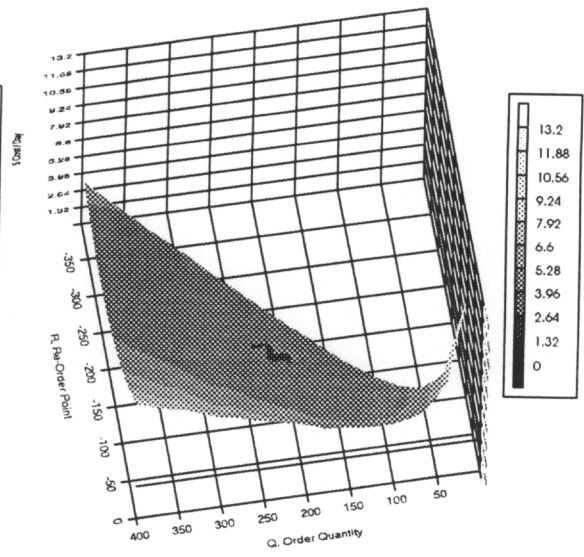
Figure 3.4 shows 3-dimensional plots of the mean of the response for all 210 points in the regions -- an exhaustive covering of the decision space indicating a good representation of the true surface, even for the worst case (HMW). However, as contour plots of the mean (Figure 3.5) show, there is a marked decrease in the accuracy of the representation if there are not enough replications. Distortion is possible even in this relatively simple inventory model. For example, case HMW with two replications suggests that the true response surface might be multimodal, which it is not.

Standard Deviation

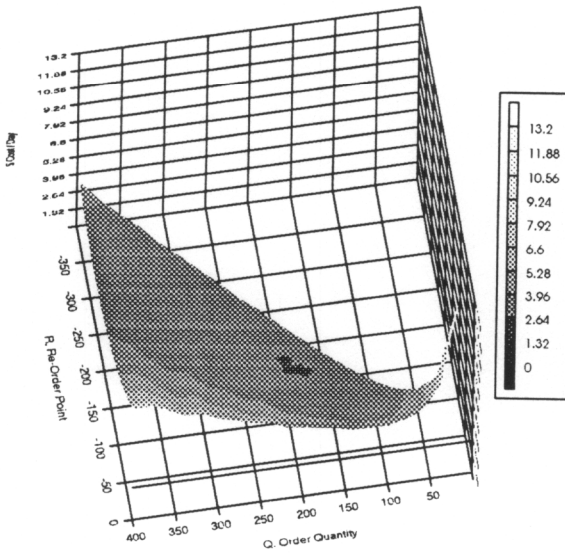
Figure 3.6 shows the plots of the standard deviation of the response. The effect of design conditions and inherent variability in the system being modeled begin to appear when one examines the standard deviation (SD) at each gridpoint in the region. SDs are quite consistent for the first case, LNB (Figure 3.7), where all are below 0.3. Although some SDs are doubled in the next two cases (LNW and HNB), the HMW case shows marked degradation in consistency (less than half of the SD's are below 0.3 and approximately 12% are above 0.6). This again reinforces the notion that a simple model can have problems with the representation of the actual surface. Furthermore, these results raise concerns that homogeneity of variance assumptions may not hold for statistical tests (see section on homogeneity of variance).



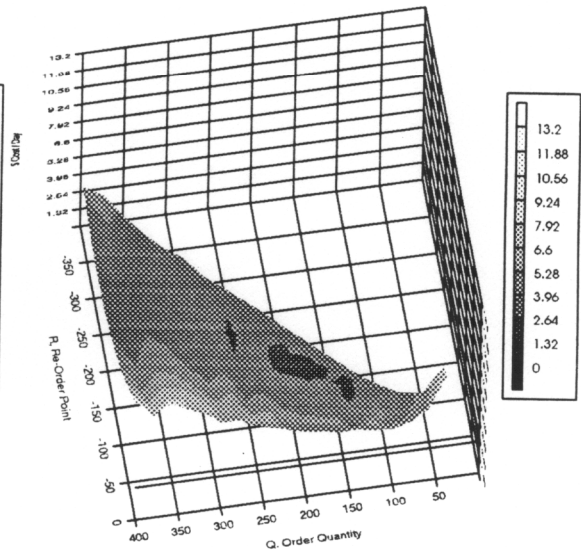
Case LNB



Case HNB

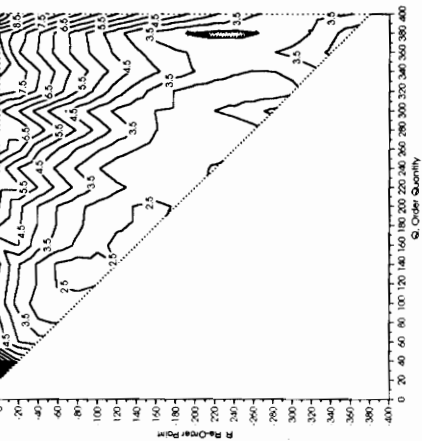


Case LNW

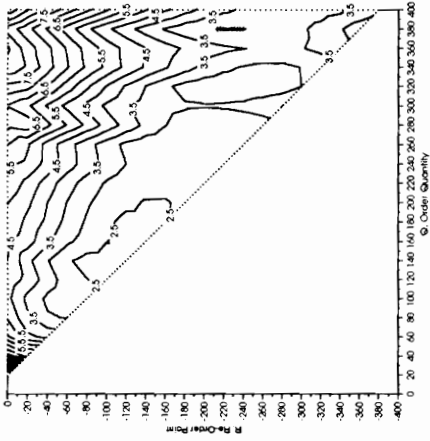


Case HMW

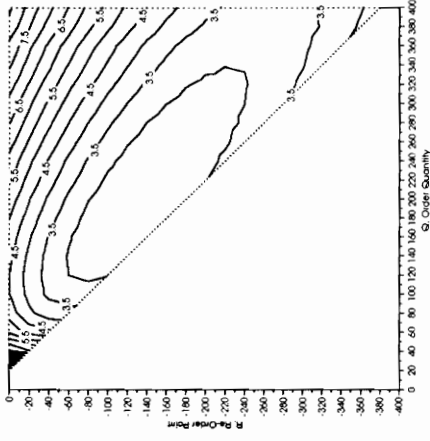
Figure 3.4. Three-dimensional plots of the mean of the response



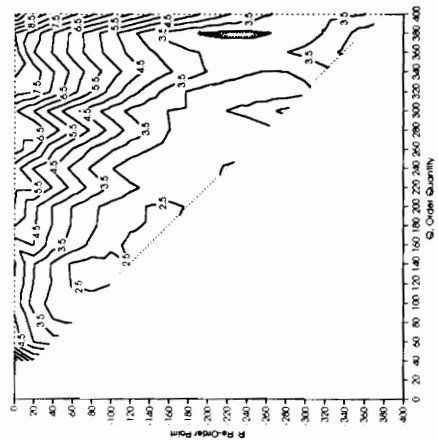
Case LNB - One replication



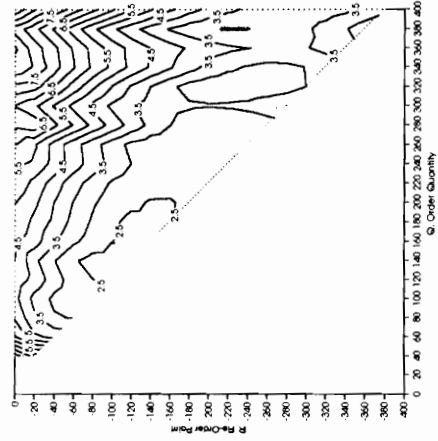
Case LNB - Two replications



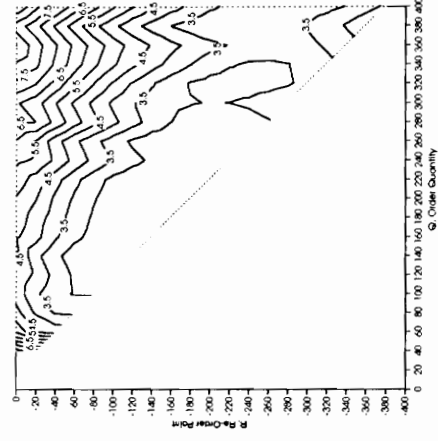
Case LNB - Three replications



Case HMW - One replication

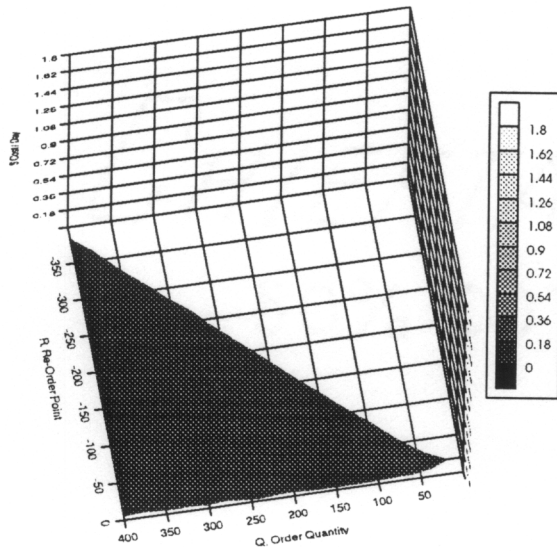


Case HMW - Two replications

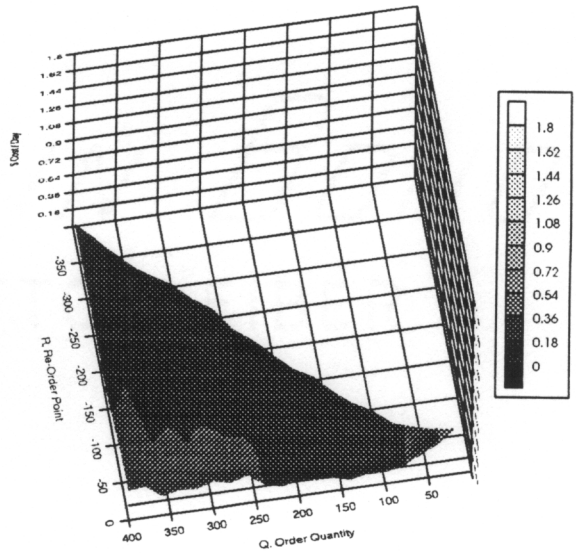


Case HMW - Three replications

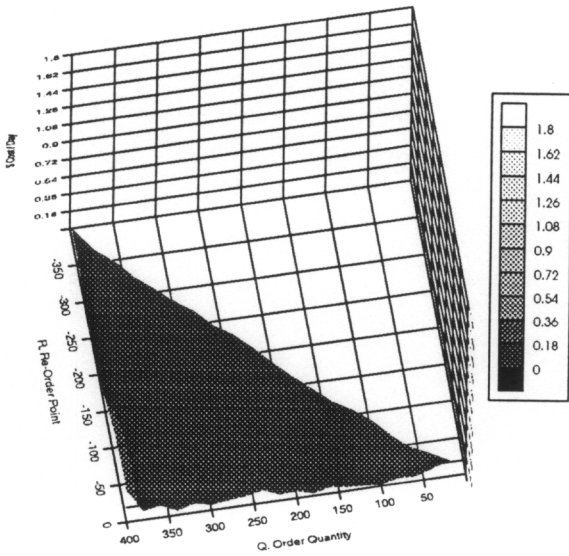
Figure 3.5. Contour plots of the mean of the response for Cases LNB and HMW



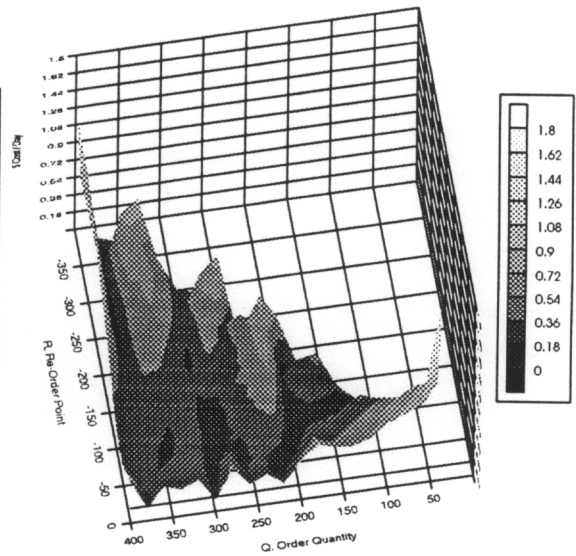
Case LNB



Case HNB

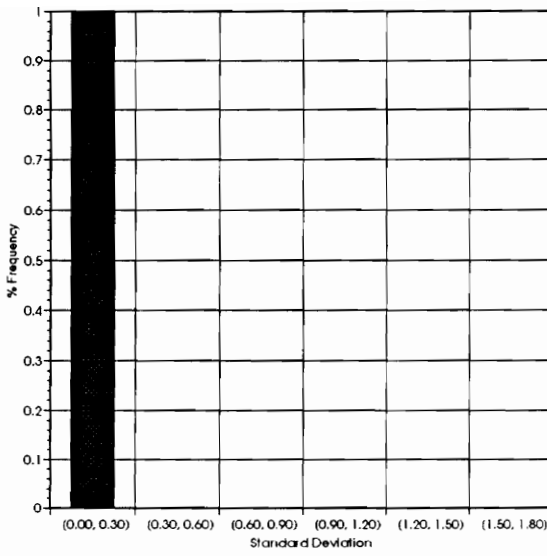


Case LNW

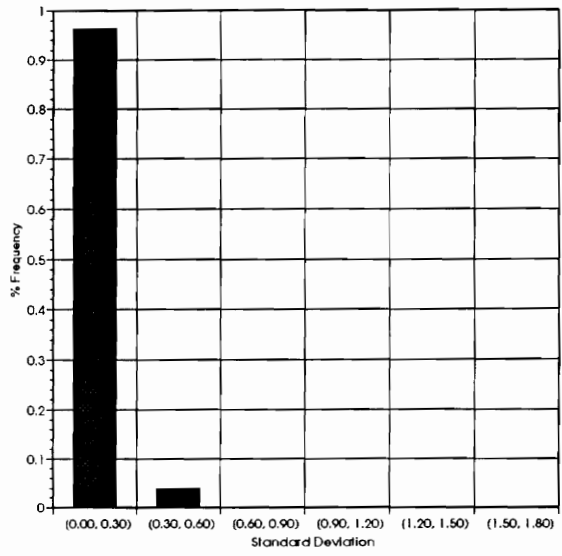


Case HMW

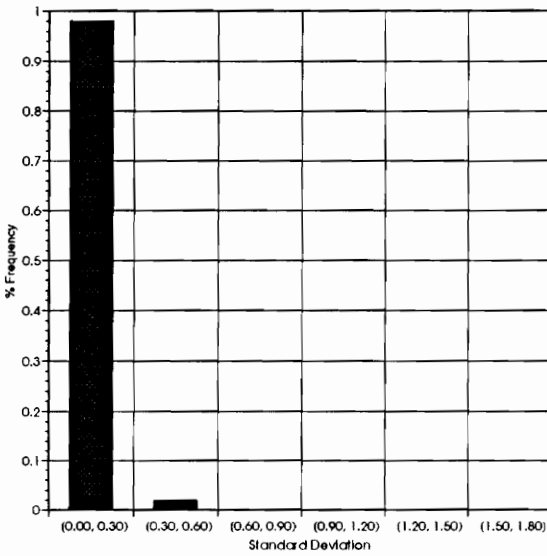
Figure 3.6. Three-dimensional plots of the standard deviation of the response



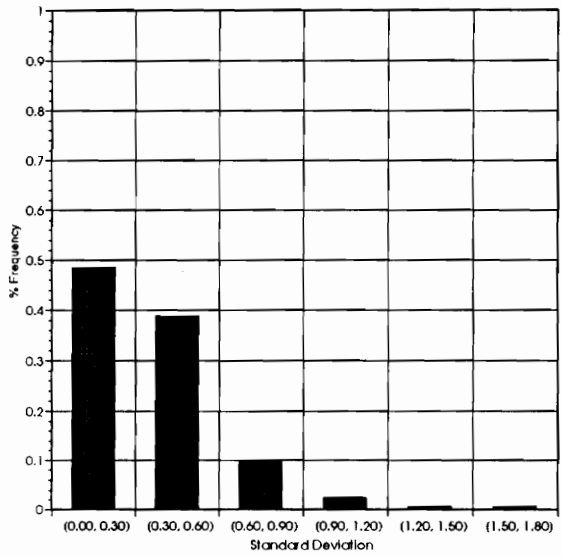
Case LNB



Case HNB



Case LNW



Case HMW

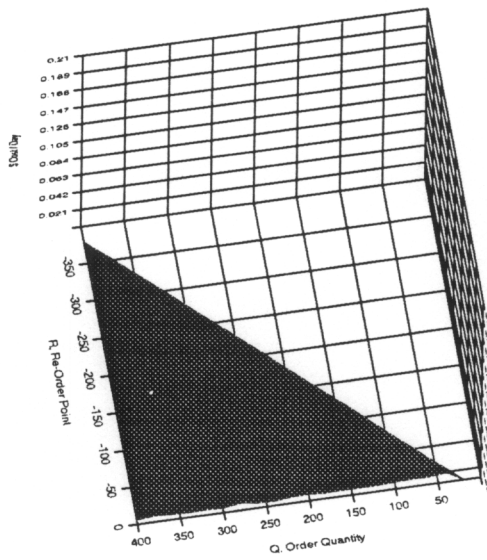
Figure 3.7. Histograms of the standard deviation of the response

Coefficient of Variation.

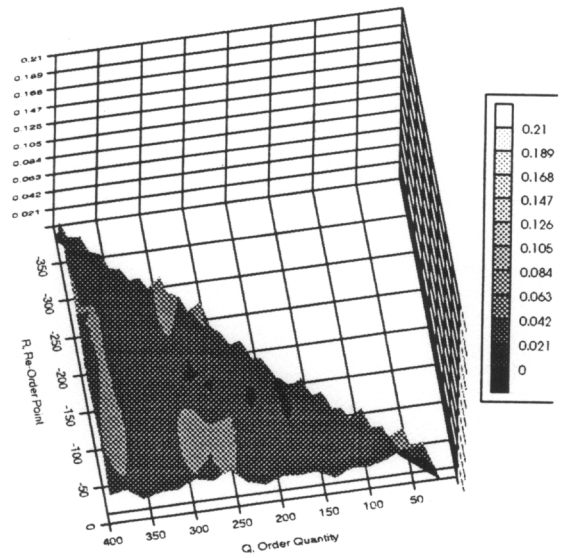
Measures that incorporate both the mean and the standard deviation are also revealing. The coefficient of variation of the response, CV, is one such measure, and is defined in (10) above. Figures 3.8 and 3.9 show 3-D plots and histograms of the CV. First note that the simulation model dampens the relative variability. E.g., the CV of the interarrival time of demand for Case LNB is one (1.0) while the largest CV of the response is less than 0.045 (less than 5% of the variability of the demand.) The simulation model has attenuated the demand variability by a factor of 20. Even in the extreme case, HMW, the CV of the interarrival time of demand is 4.8 but the CV of the response is less than 0.225, again an attenuation of about the same factor. As shown in Figure 3.9, Case LNB shows low relative variability across the entire surface (less than 0.045); case HNB's surface has more relative variability (12% of the CV's are above 0.045); case HMW shows considerable variability with only 28% of the CV's below 0.045, over 40% above 0.09, and approximately 5% above 0.18. These results are consistent with the observations made to this point. The true response at each point does not change, but the variability about that value is increasing, although not evenly across the surface.

Signal to Noise Ratio

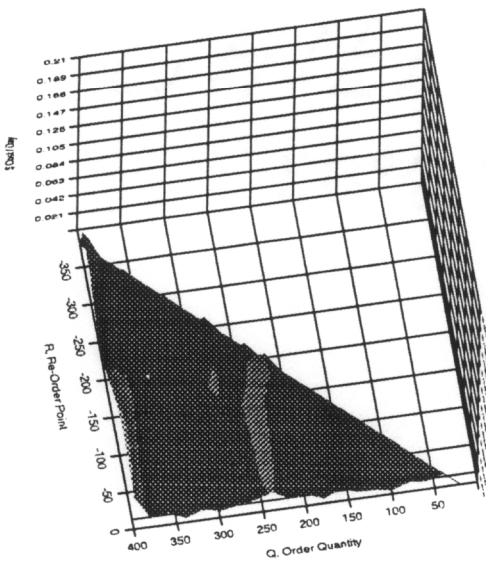
The point measure defined in (Keys, Rees, Greenwood, 1995a) above, the signal-to-noise ratio (S/N), is similar to CV but is logarithmic; with this measure, a high S/N is preferred since this indicates relatively low error. The S/N plots (Figures 3.10 and 3.11) are consistent with the CV plots. The S/N's for Case LNB are the most consistent with over 96% of the ratio between 37.5 and 46.0 (Figure 3.11). The uncertainty in the process becomes more evident in the other cases as less than half of the S/N values are above 37.5 in Case LNW (Figure 3.11) and almost none are above 37.5 for cases HNB and HMW. In fact,



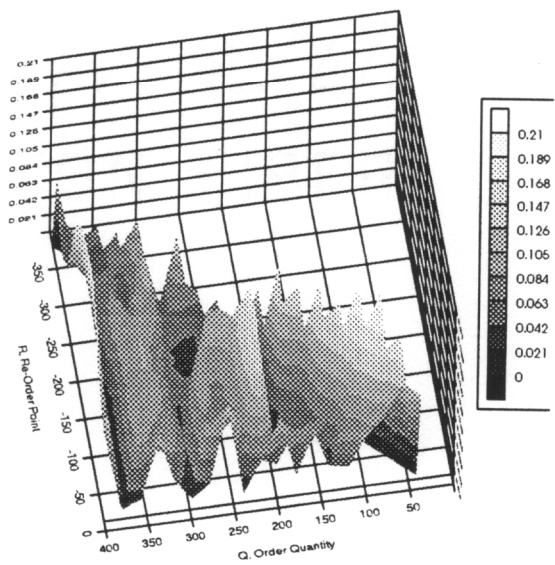
Case LNB



Case HNB

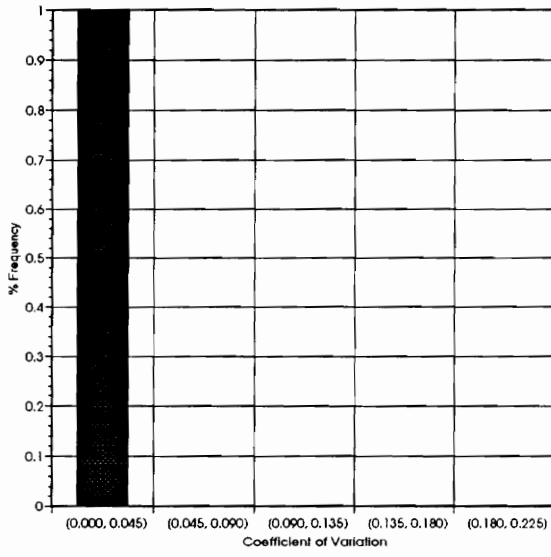


Case LNW

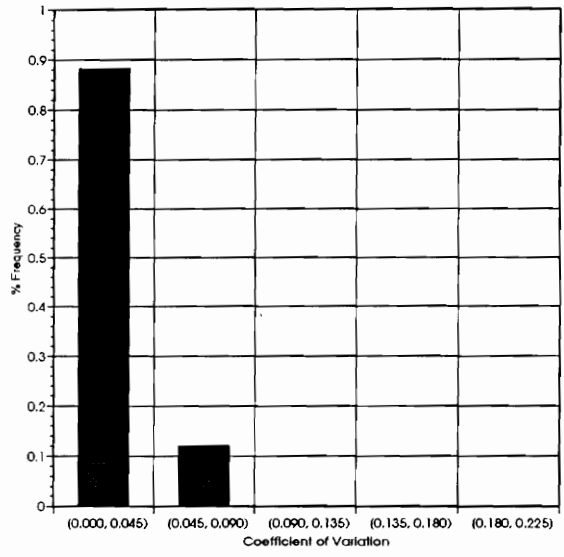


Case HMW

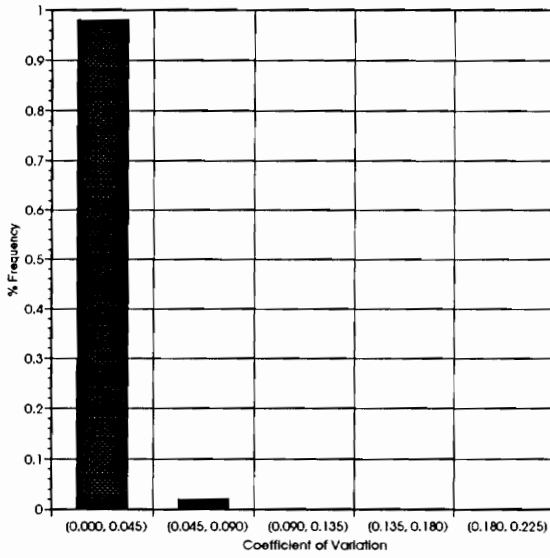
Figure 3.8. Three-dimensional plots of the coefficient of variation of the response



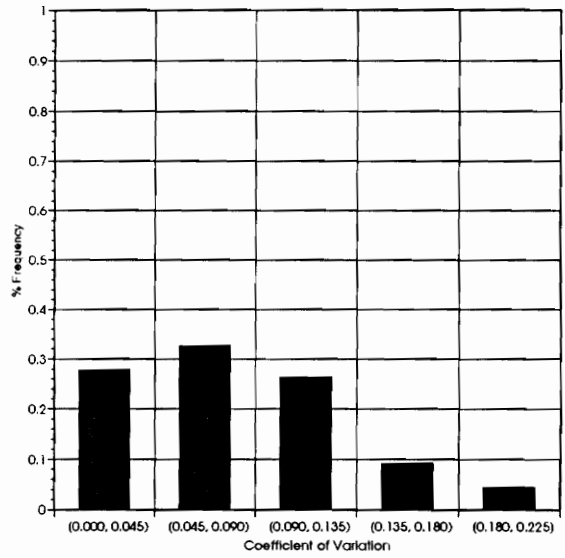
Case LNB



Case HNB

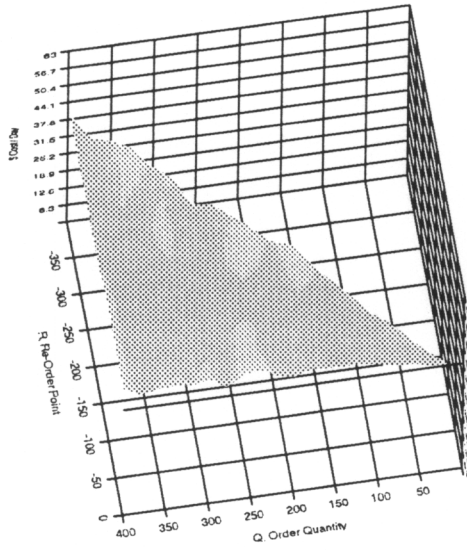


Case LNW

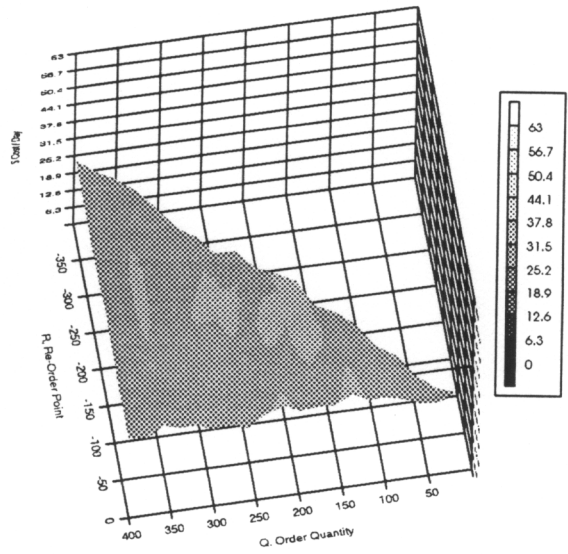


Case HMW

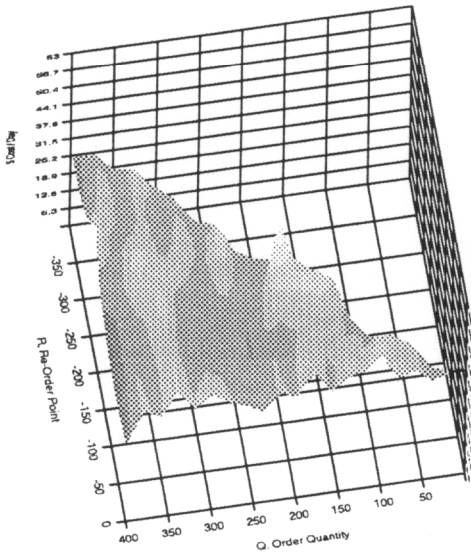
Figure 3.9. Histograms of the coefficient of variation of the response



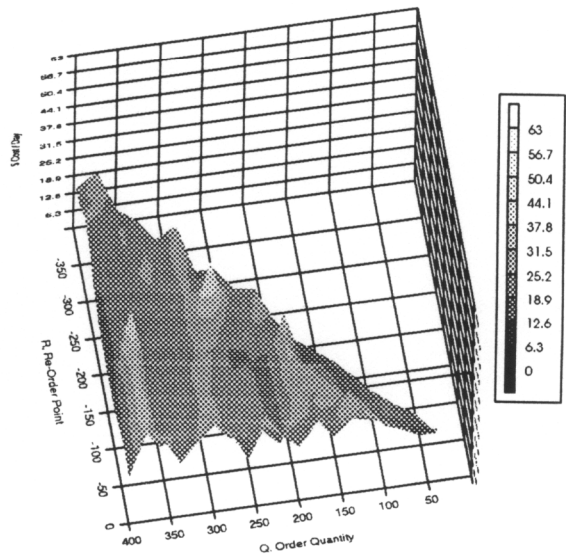
Case LNB



Case HNB

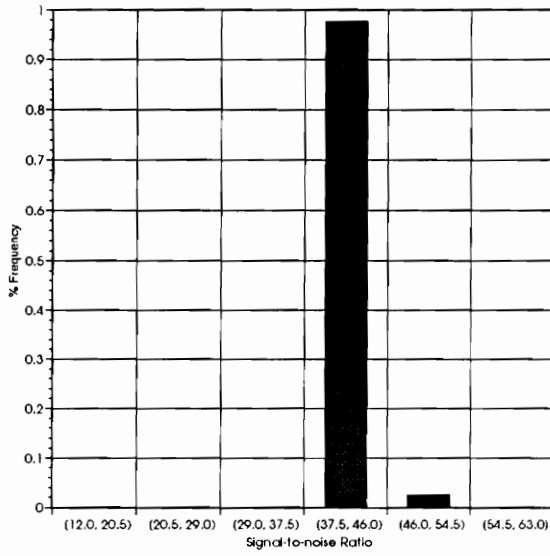


Case LNW

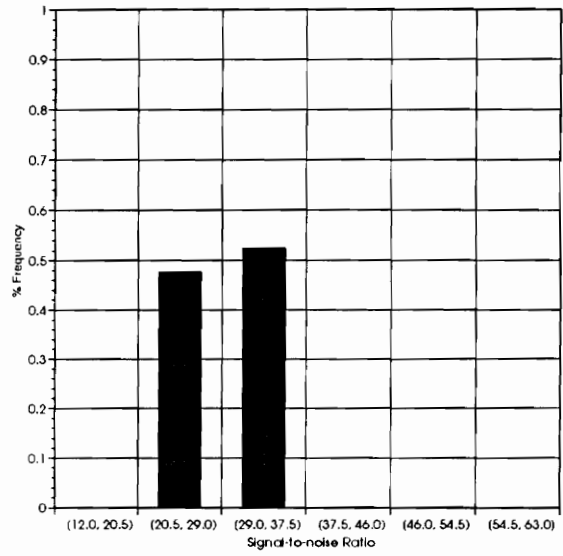


Case HMW

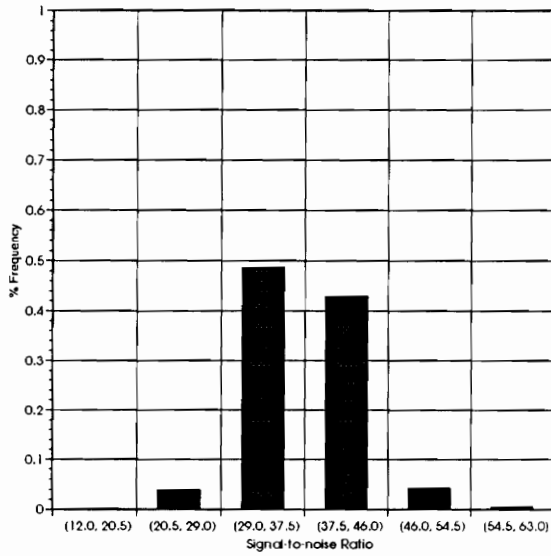
Figure 3.10. Three-dimensional plots of the signal-to-noise ratio of the response



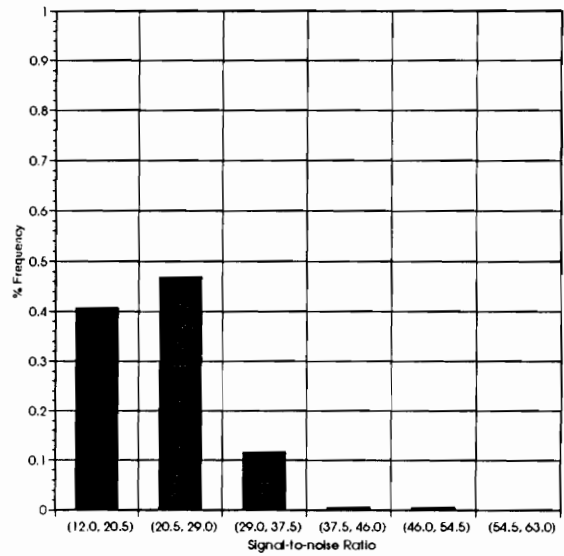
Case LNB



Case HNB



Case LNW



Case HMW

Figure 3.11. Histograms of the signal-to-noise ratio of the response

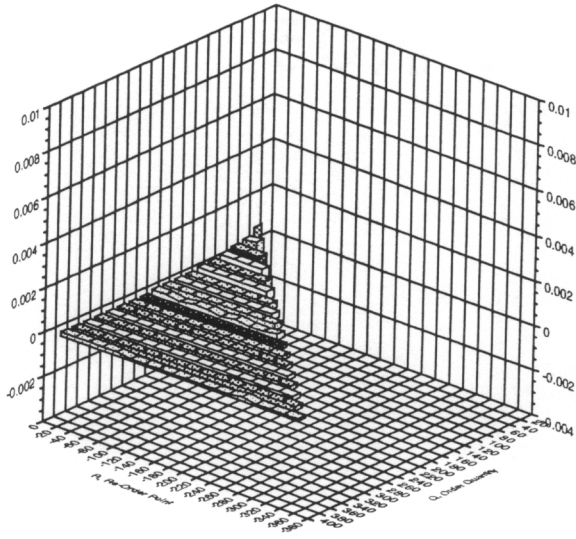
nearly half of the S/N ratios are below 29.0 in Case HNB (Figure 3.11) and nearly 90% below 29.0 in Case HMW (Figure 3.11).

A comparison of Cases LNB and LNW illustrate the effect of the design conditions -- run length and replications. Fewer replications and shorter runs add to the lack of consistency in the S/N ratio and also produce more noise. Comparing Cases LNW and HMW reveals that greater inherent model variability also produces more noise, but not necessarily greater spread in the S/N ratio. This suggests that for a given simulation model, the developer's choice of run length and replications can significantly affect the amount of noise in the system and hence the S/N, CV and SD. This indirectly affects the validity of statistical tests and the appropriateness and efficacy of different search methods.

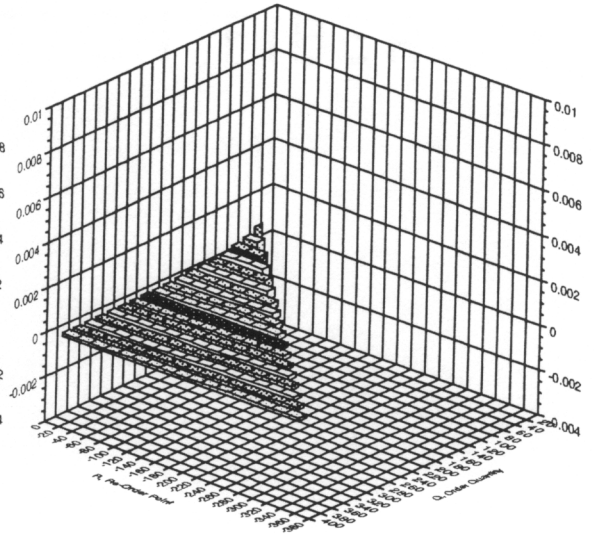
Relative Activity or “Bumpiness.”

Figure 3.12 shows that with the decision variable Q held constant there is very little activity as all plots are almost completely flat or horizontal. However, the plots of bumpiness with the decision variable R held constant (Figure 3.13, note the change in the axes) show a lot more activity as well as some acceleration; i.e., the system is more sensitive to the variable Q than it is to R. That this is so may also be seen from Figure 3.4 which shows much more response variability as Q is changed than when R is varied. The plots in Figures 3.12 and 3.13 also indicate that the higher the inherent system variability, the more bumpiness there is.

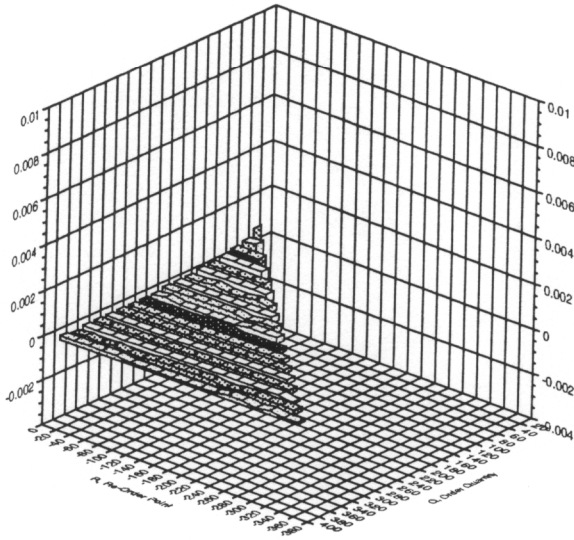
Lower values for bumpiness in a region should indicate higher confidence that the region is well understood. Conversely, higher values indicate lower confidence and can therefore suggest to the modeler areas where additional experimentation may be helpful or necessary. Design conditions and inherent model variability both affect bumpiness. Increasing the number of replications and the duration of the runs results in lower values for bumpiness. The plots indicate that the greater the inherent model variability the greater the values of bumpiness. For the cases considered, inherent model variability has a



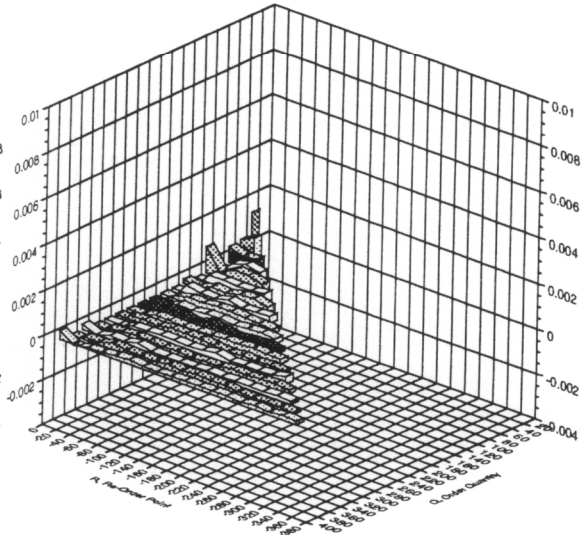
Case LNB



Case HNB

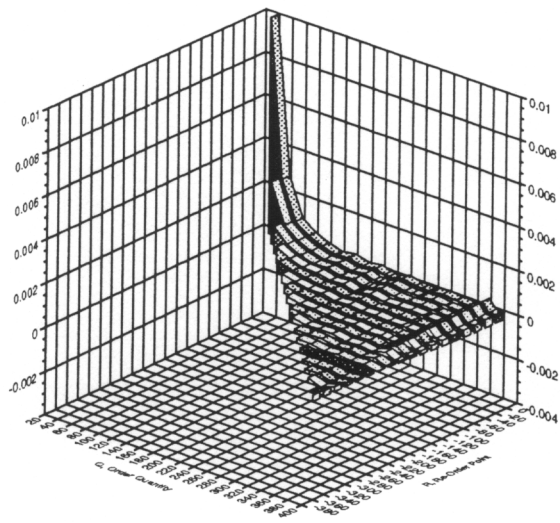


Case LNW

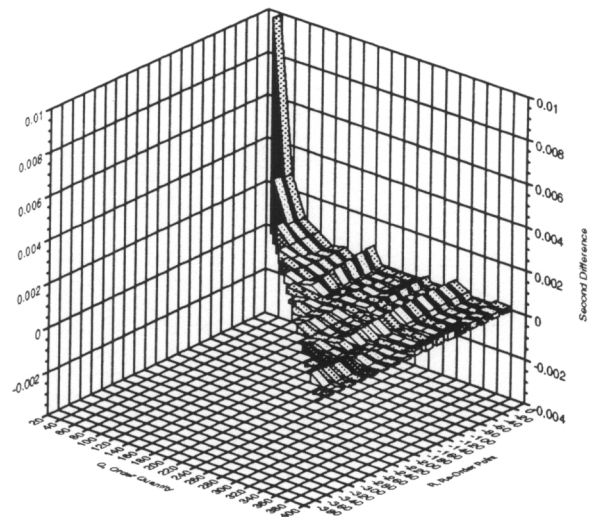


Case HMW

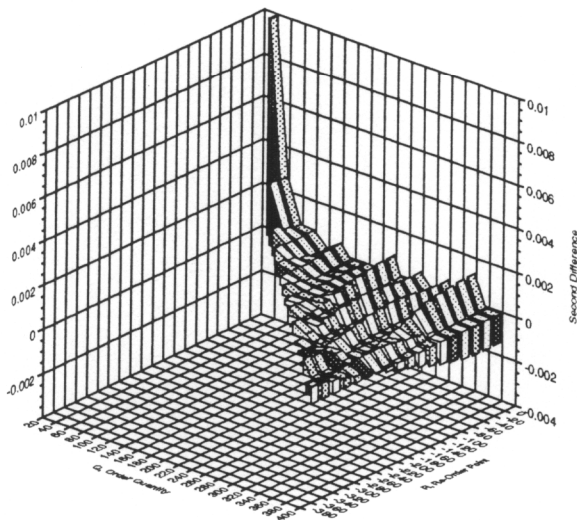
Figure 3.12. Second differences of the response with Q held constant



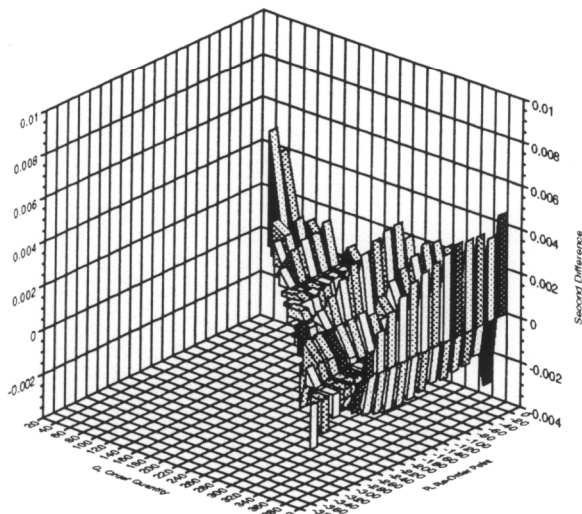
Case LNB



Case HNB



Case LNW



Case HMW

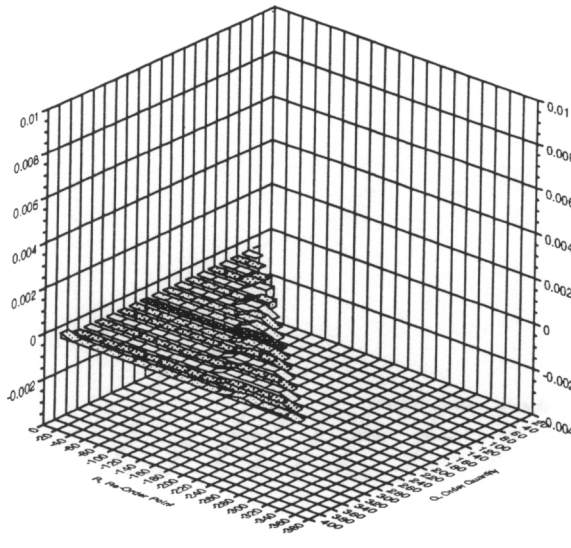
Figure 3.13. Second differences of the response with R held constant

lesser impact on bumpiness than do design conditions. The plots showing second differences when the first difference is approximately zero (Figures 3.14 and 3.15) point out local optima. When the second difference is positive the local optimum is a minimum, the case of interest for the inventory model. Note first in these two figures that all second differences are very small, suggesting very little acceleration/deceleration in the surface; this is consistent with the gently sloping nature of this simple inventory model. The plots also indicate a series of local minima close to the $Q = |R|$ edge of the feasible region of the surface. The locations of the local minima are consistent with the plots of the means and include the “true” minimum. Figure 3.15 case HMW also shows some activity near the line $Q = 380$, but further investigation shows that this is just a local optimum due to a ripple in the response surface.

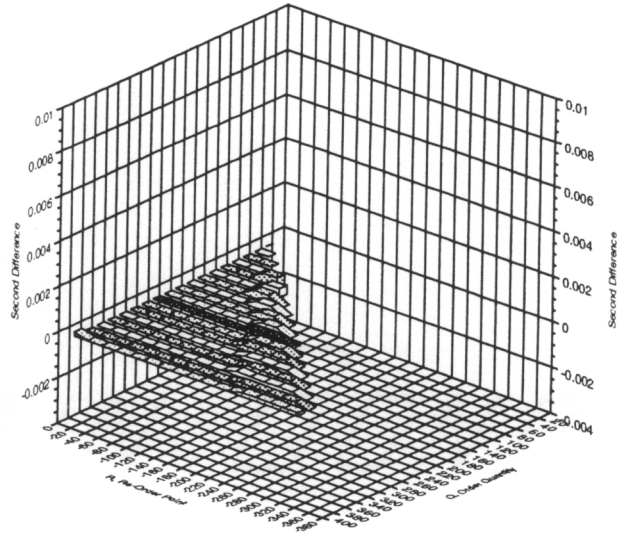
REGIONAL ESTIMATES

Test for Normality

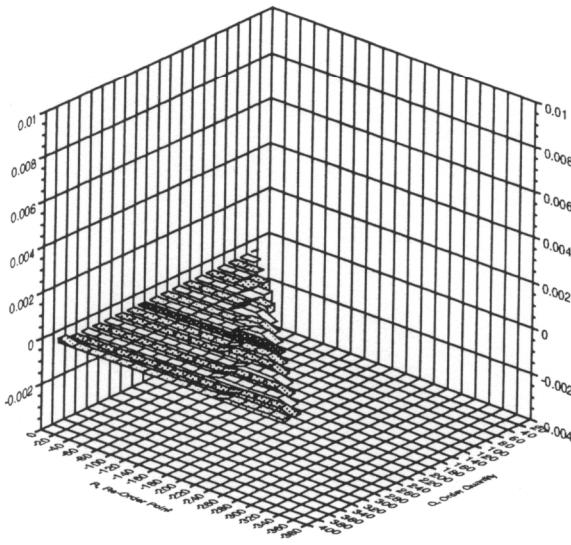
Figures 3.16 and 3.17 show three-dimensional plots of p-values for the Shapiro-Wilk (S-W) test of normality, with one p-value calculated and plotted for each region. Figure 3.16 reports the p-values for the $\Delta =$ inter-gridpoint spacing = 20 case, whereas Figure 3.17 shows the same for the $\Delta = 40$ case. Recall that for the S-W test, a p-value lower than the α -level indicates that the normality assumption must be rejected, whereas higher (than α) p-values imply there is less evidence against the null hypothesis that the distribution is normal. Hence, in Figure 3.16, approximately only 5% (Case LNB) to 13% (Case HMW) of the regions fail to meet the normality assumption at an $\alpha=0.05$ level. For Cases LNB and LNW, there is very little difference in the distribution of p-values over the region, as indicated by the histograms in Figure 3.16. This implies design conditions have little impact on the normality assumption for the model and levels considered. Inherent variability in the system appears to have more of an effect,



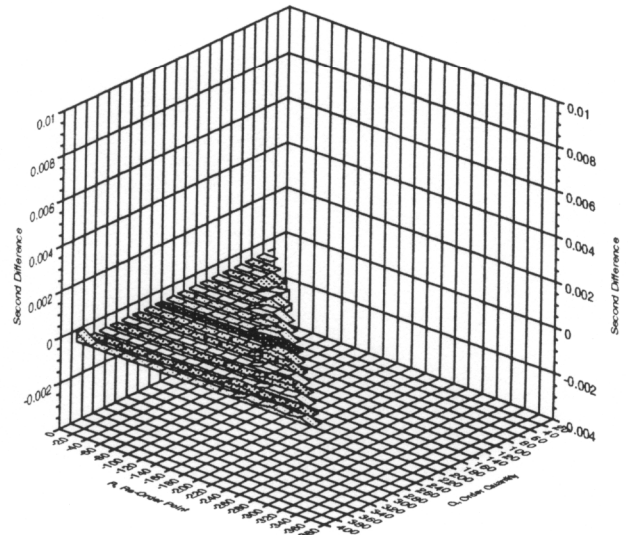
Case LNB



Case HNB

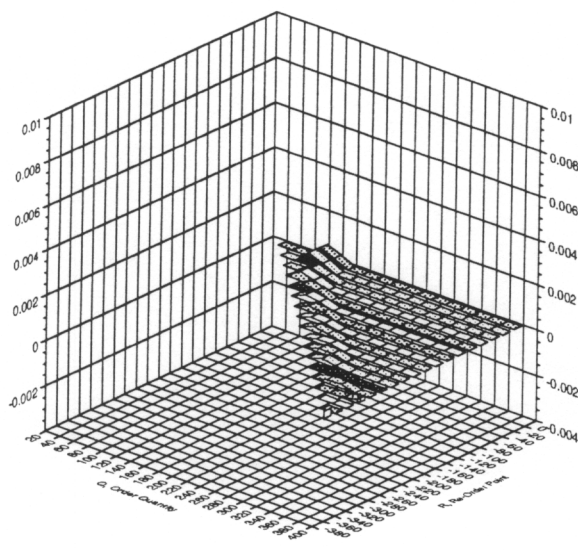


Case LNW

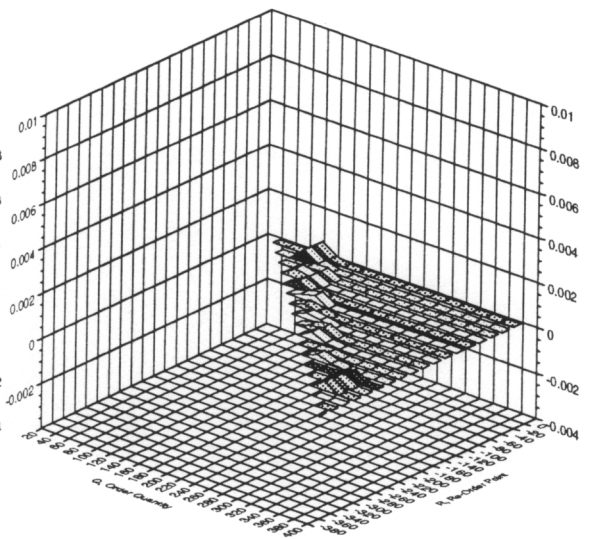


Case HMW

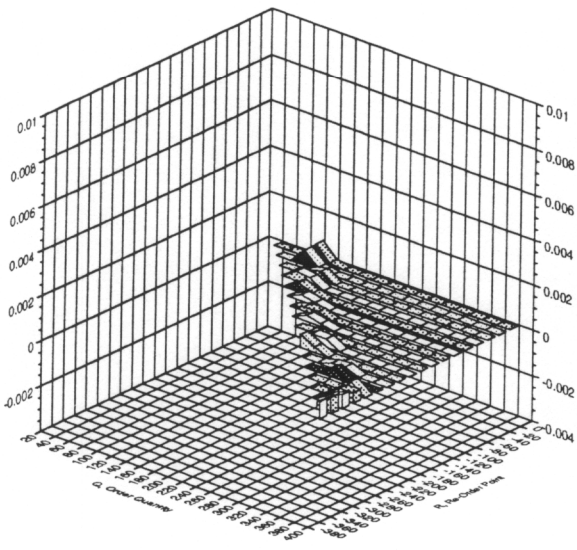
Figure 3.14. Second differences of the response when the first difference crosses zero; with Q held constant



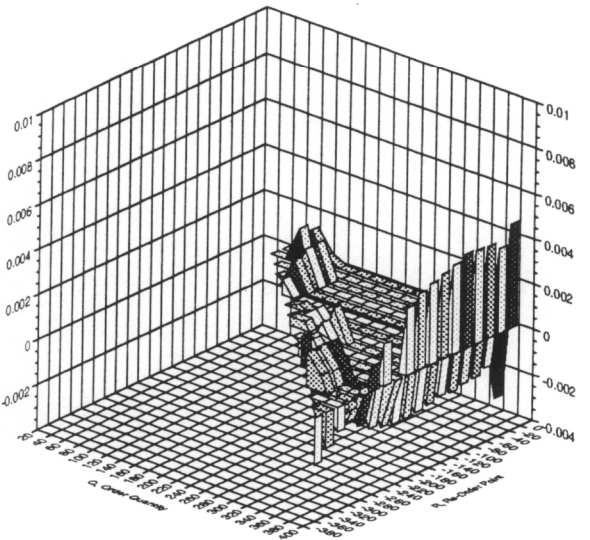
Case LNB



Case HNB



Case LNW



Case HMW

Figure 3.15. Second differences of the response when the first difference crosses zero; with R held constant

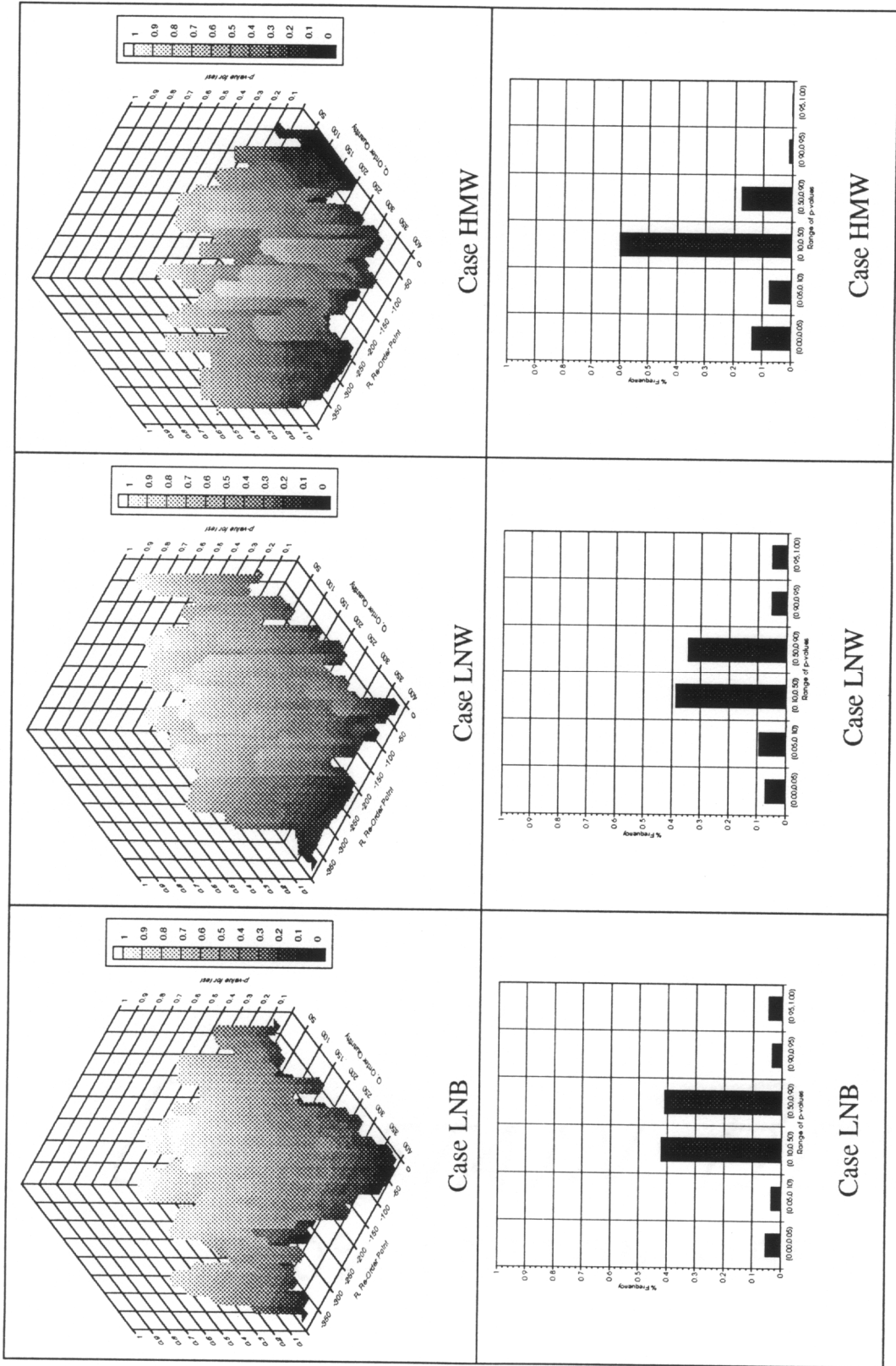


Figure 3.16. Three-dimensional plots and histograms of p-values for Normality Test, $\Delta=20$

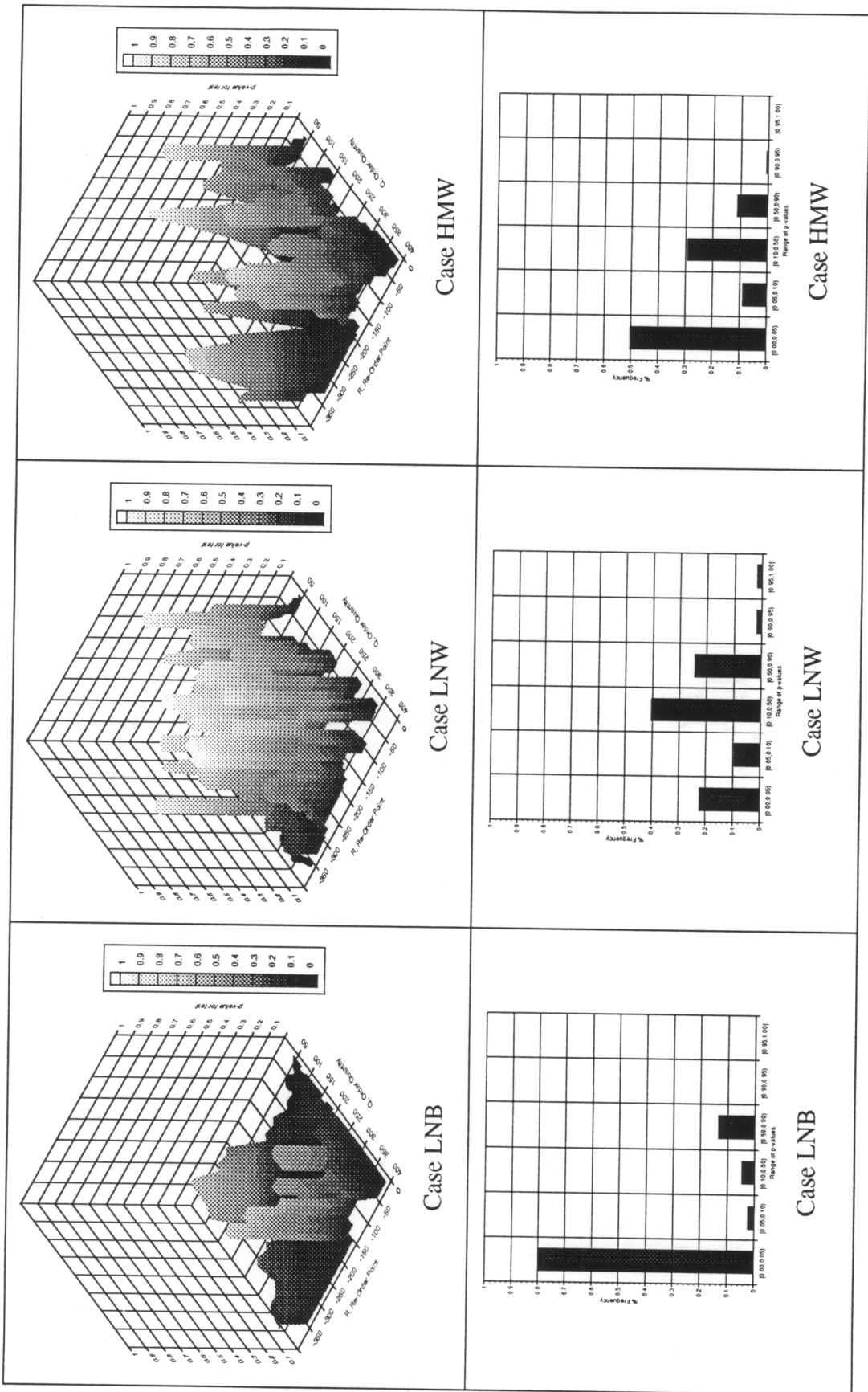


Figure 3.17. Three-dimensional plots and histograms of p-values for Normality Test, $\Delta=40$

as seen in the comparison of Cases LNW and HMW. Based on the S-W test defined above, case HMW provides stronger evidence against the null hypothesis, which states that the data follows a normal distribution. This finding implies that increased variability in the system makes it more difficult to meet normality assumptions required in many statistical tests.

The most noticeable and important finding from Figure 3.17, which reports on the $\Delta = 40$ spacing, is that all three histograms shown indicate a much greater percentage of regions that have non-normally distributed residuals. The range of percentages of regions not meeting normality ranges from 22% (case LNW) to 80% (case LNB). These results compare with a range of 5% to 13% for the $\Delta = 20$ case. This indicates that the choice of inter-gridpoint spacing greatly affects the validity of F-tests used commonly in such simulation-optimization techniques as RSM.

The explanation for the increased rejection of normality for $\Delta = 40$ is as follows. With $\Delta = 20$, possible region configurations are \mathcal{R}_3 and \mathcal{R}_4 . Each of these configurations contains gridpoints only on the boundary of the region. Conversely, with $\Delta = 40$, possible region configurations ($\mathcal{R}_6, \mathcal{R}_8, \mathcal{R}_9$) contain points both on the boundary and the interior of the region. What is typically happening in the regions that fail to have normal residuals is that the hyperplane is not fitting the data well due to curvature. For example, the hyperplane may overestimate the responses on one region boundary, underfit it for the interior points, and fit fairly well through the opposite boundary. This leads to a distribution of residuals that is highly non-normal. A plot showing one such histogram of residuals for the ten replications across the \mathcal{R}_6 region defined by $\mathcal{R} = \{(Q, R) = (20, 0); (40, 0); (40, -20); (60, 0); (60, -20); (60, -40)\}$ is shown in Figure 3.18a; precisely what was described about curvature has happened in this region. The ten residuals in the right-most rectangle in the figure all come from the point (20, 0); the twenty residuals in the left most rectangle in Figure 3.18a all come from the points (40, 0) and (40, -20). The plane overestimated the responses at the left of the region, underestimated those in the interior, and fit those on the right boundary better. These residuals do not follow a normal distribution, as may be seen by inspection of

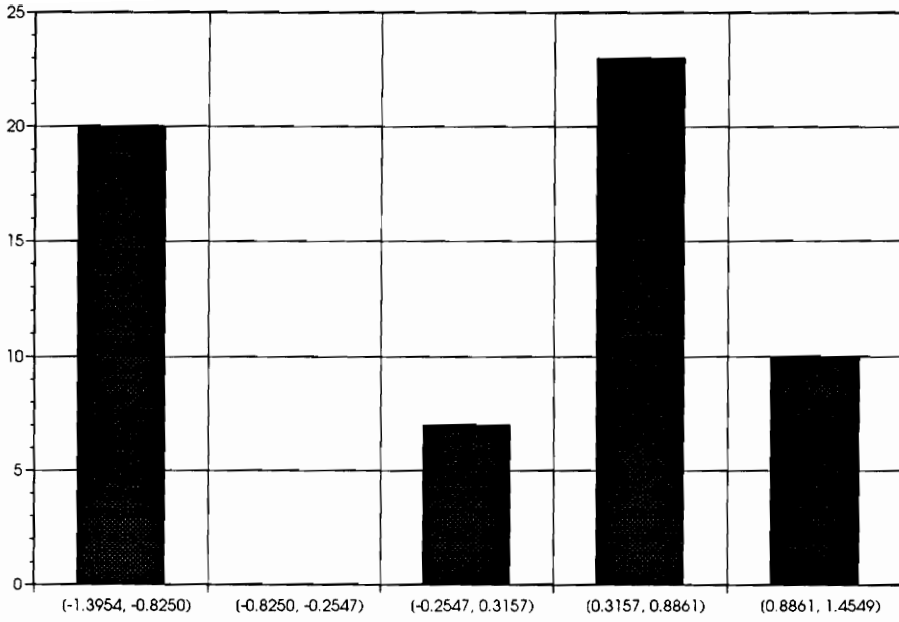


Figure 3.18a. The distribution of residuals for a “typical” \mathcal{R}_6 ($\Delta=40$)

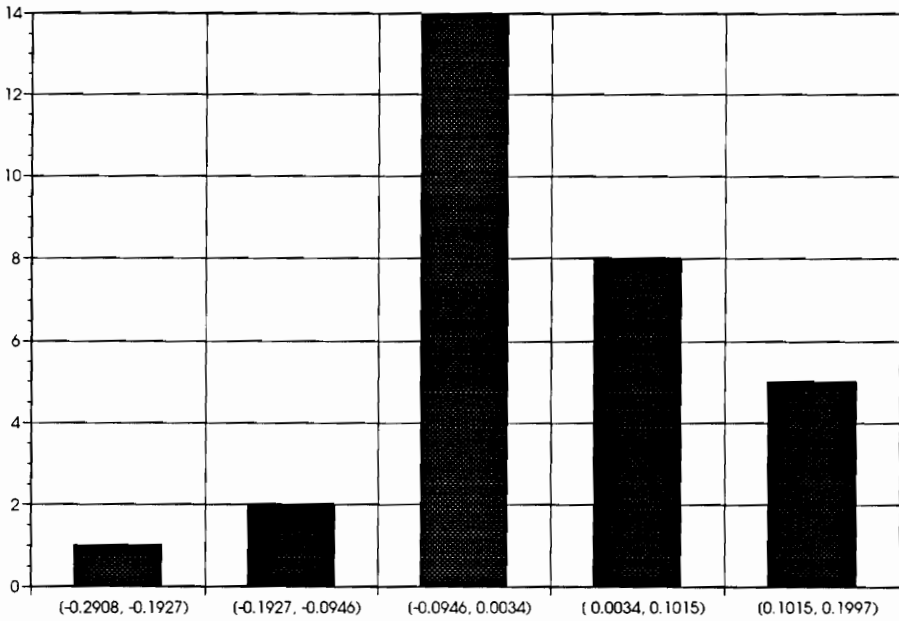


Figure 3.18b. The distribution of residuals for a “typical” \mathcal{R}_3 ($\Delta=20$)

Figure 3.18. The distribution of residuals for different inter-gridpoint spacings.

Figure 3.18a. Conversely, the residuals for the $\Delta = 20 \mathcal{R}_3$ region case defined by $\mathcal{R} = \{(Q, R) = (20, 0); (40, 0); (40, -20)\}$ are shown in Figure 3.18b and are much more normal in appearance as the plane gives a much more even fit with no interior points. (The S-W test fails to reject normality.)

Note that it is inappropriate to perform a parametric first-order lack-of-fit test over the \mathcal{R}_6 described above, as such a test requires normality. It is possible that naive RSM users conduct such tests incorrectly as the RSM procedure is sometimes explained without inclusion of steps checking assumptions (such as normality).

Homogeneity of Variance

Plots for homogeneity of variance in each region are shown in Figures 3.19 and 3.20. Recall that in these figures p-values above 0.05 suggest homogeneous variances in the region (at $\alpha=0.05$), whereas values below 0.05 indicate heterogeneity.

These plots show that, in general, variances are not heterogeneous for either $\Delta = 20$ or $\Delta = 40$, regardless of the design conditions. Even the HMW cases show that only 5% or 6% of the regions fail to possess homogeneity tests. Of course the simulation model studied in this research is fairly straightforward, and other more complex models might lead to increased heterogeneity. But for the conditions studied here, the homogeneity assumption required for F-tests is rarely violated.

“SEARCHABLE” REGIONS

This section addresses the question of whether search techniques using first-order metamodels to determine gradient-search directions, such as RSM, can be properly used over the various regions \mathcal{R}

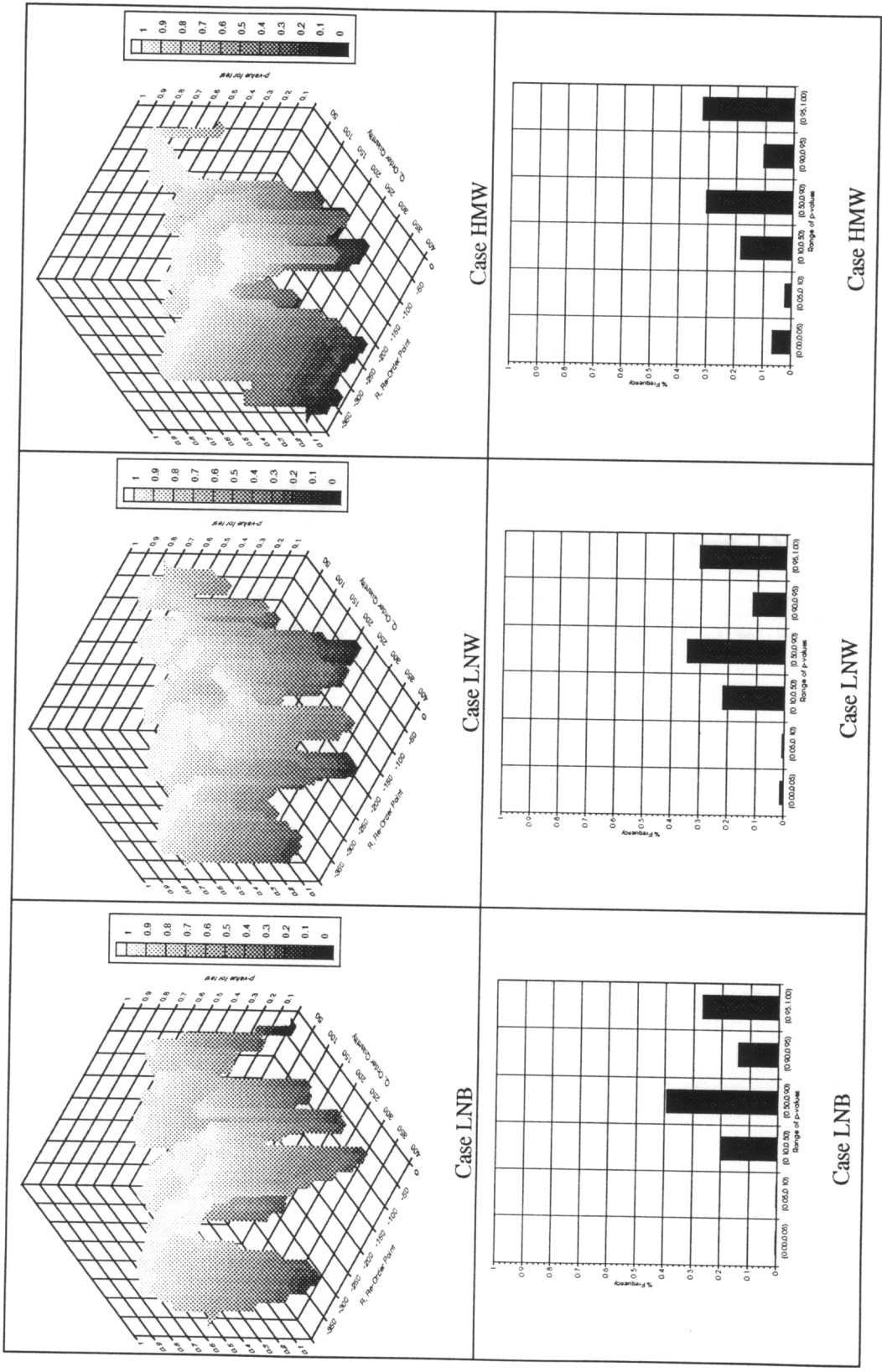


Figure 3.19. Three-dimensional plots and histograms of p-values for Homogeneity-of-Variance Test, $\Delta=20$

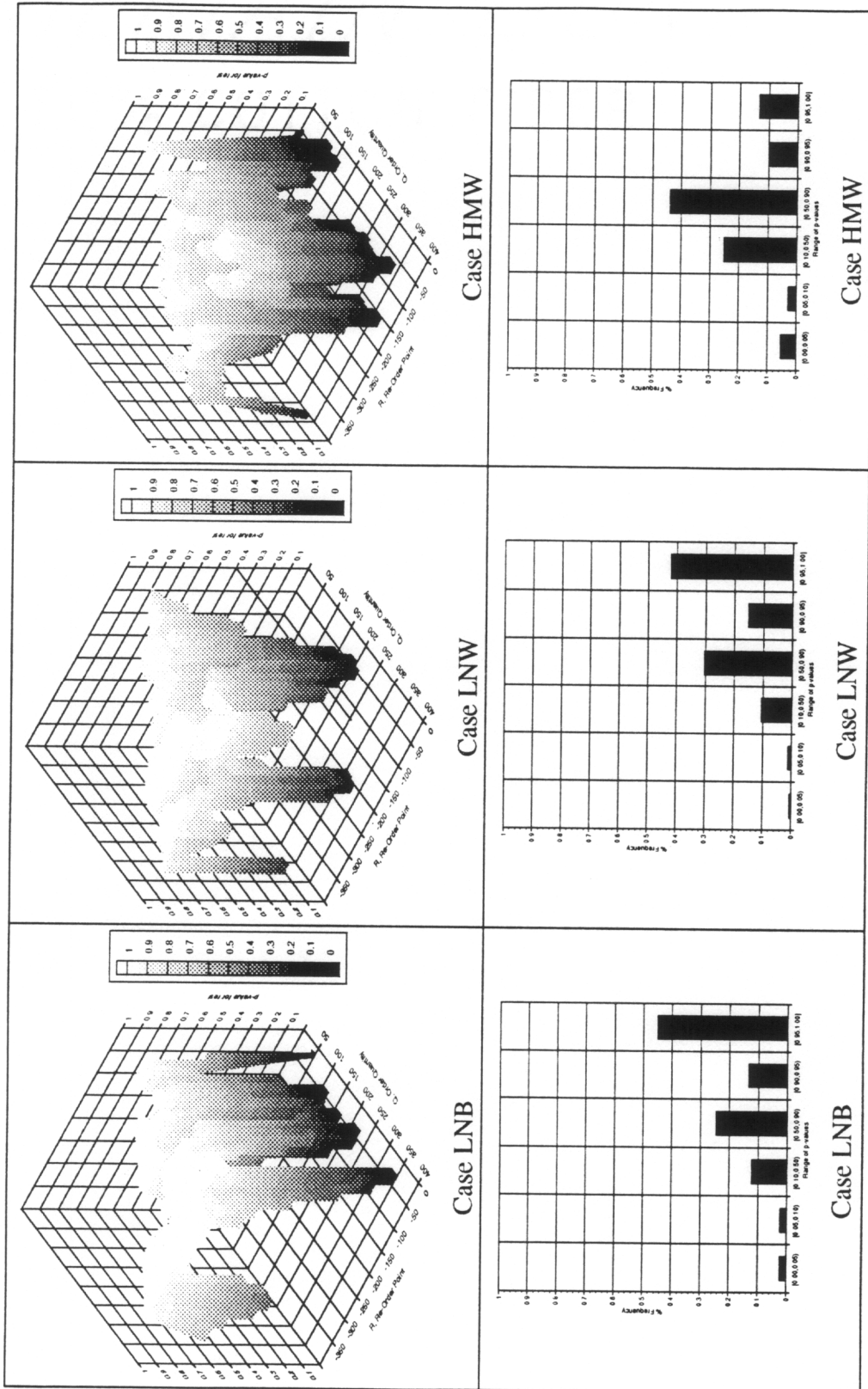


Figure 3.20. Three-dimensional plots and histograms of p-values for Homogeneity-of-Variance Test, $\Delta=40$

comprising the feasible region. Two inter-gridpoint spacing cases, $\Delta = 20$ and $\Delta = 40$ are examined, and each is further studied on four experimental design cases (LNB, LNW, HMB, HMW).

The issue of whether first-order RSM models can be properly invoked is important because we will show that even with this simple, inventory simulation model, traditional RSM is not appropriate for many regions \mathcal{R} . The implications of this finding will be discussed in the Conclusions section.

As mentioned, in order to fit a linear, first-order model over a region and ensure adequate, non-horizontal fit, two F-tests must be passed: the significance-of-regression test and the lack-of-fit test. These tests in turn require that the region consists of normally distributed residual errors and homogeneous variance over the region. Therefore, strictly speaking, for a region to have a first-order model properly fit and be a candidate for RSM's gradient search, the following four conditions should hold:

- (1) the region passes the Shapiro-Wilk normality test ($p > 0.05$);
- (2) the region passes the Bartlett test for homogeneity of variance ($p > 0.05$);
- (3) the region passes the significance-of-regression test ($p < 0.05$); and
- (4) the region fails the lack-of-fit test, i.e., lack-of-fit fails to be true ($p > 0.05$).

Figures 3.21 and 3.22 are constructed to indicate which areas are "searchable" with first-order RSM metamodels. These figures show the four possibilities:

- (1) violation. A region so classified has failed either the normality or the homogeneity of variance test. In Figure 3.21, regions in this category are shaded in black.
- (2) flat. A region classified in this grouping has failed the significance-of-regression test. In Figure 3.21, these regions are shaded dark gray.

- (3) first order. A region in this category is a candidate for RSM's gradient search and has met all four of the conditions stated above. These regions are shaded light gray in Figure 3.21.
- (4) second-order/other. These regions, which are left white in Figure 3.21, have passed the normality, homogeneity-of-variance, and significance-of-regression tests. But the regions contain significant lack-of-fit, suggesting that a second-order or some other model is needed to adequately fit the response over the region; a first-order model is inadequate.

The first row of panels in Figures 3.21 and 3.22 describes results for the $\Delta = 20$ inter-gridpoint spacing case. Observe that many areas are "RSM-able" (i.e., light gray). The first panel in the first row (case LNB) contains many regions (about 68%) that are white and therefore require a second-order/other metamodel. This finding is somewhat in opposition to the popular RSM strategy of fitting a first-order model initially when one is probably far from the true optimum. Case LNW is much more appropriate for an RSM strategy with about 80% of the regions light gray (i.e., first-order). As demand variability is increased (HNB) over the previous two cases, over 30% of the regions fit in a category other than first-order; about 25% violate either normality or homogeneity-of-variance requirements, and the other 8% or so are equally divided between "flat" and "second-order/other." The final $\Delta = 20$ case shows that only 31% of the regions are first-order; the rest are either in violation or have so much variance that statistically they appear flat.

The $\Delta = 40$ row of panels at the bottom of both Figures 3.21 and 3.22 indicate a much lower percentage of regions appropriate for first-order RSM. For all four cases of $\Delta = 40$, no more than 10% of the region is light gray; the other regions are either in violation of F-test assumptions (almost exclusively normality) or need non-first-order metamodels.

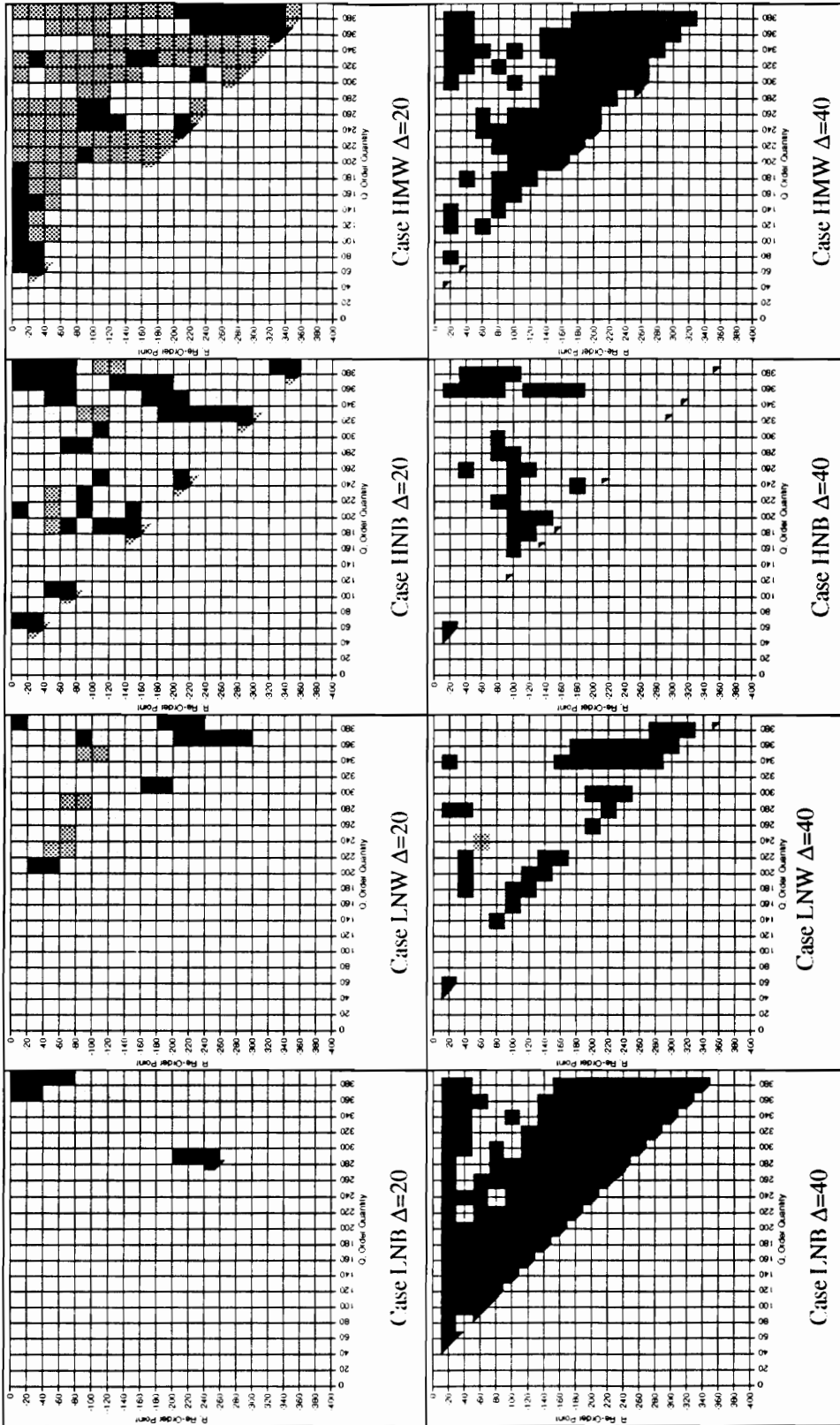


Figure 3.21. Locations of searchable areas

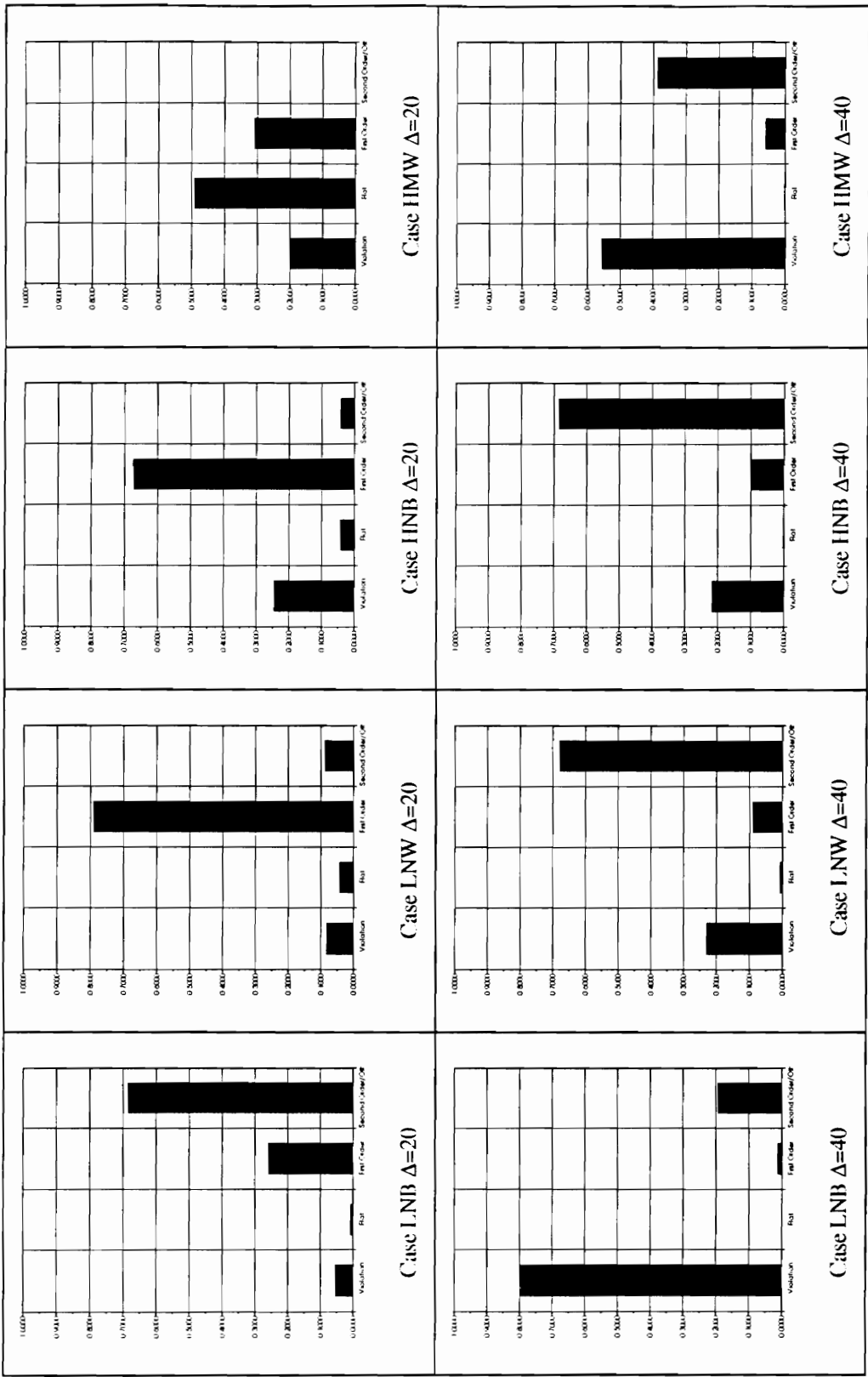


Figure 3.22. Relative numbers of searchable areas

These findings are not meant to imply that RSM is an inappropriate search strategy. Rather the issue is whether first-order models are being used appropriately within RSM and whether nonparametric tests are also needed.

CONCLUSIONS

In this study a simple, inventory-simulation model was studied under four different experimental design conditions. These conditions varied the coefficient of variance of demand and of lead time and also examined two different levels of design conditions, i.e., the number of replications and the simulation run length. A simple model was studied because it was believed that even a naive modeler intent on finding the system optimum would be able to safely and properly use a technique such as RSM, a widely used and respected approach.

The purpose of the study was to investigate common statistical measures over the search region. Both point estimates (mean, standard deviation, coefficient of variation, signal-to-noise ratio) and region measures and tests (normality of residuals, homogeneity of variance, significance-of-regression and lack-of-fit) were examined.

Point-estimate measures exhibited considerable sensitivity to experimental design conditions. This gave rise to concerns that perhaps the simple inventory model might not be simple enough to conduct

simulation-optimization searches using methods requiring some parametric statistical tests. Regional measures added some additional concerns.

That the appropriateness of various optimization approaches should be questioned was portrayed in a final set of plots indicating which points of the overall search area were amenable to first-order RSM and which were not. It was found that an important determinant of amenability was the inter-gridpoint spacing of the gridpoints. The gridpoint spacing is a very important practical issue, as one conducting optimization on a simulation model must be able to specify, e.g., in RSM, the (uncoded) size of the region of the first-order designs and the step size to be taken along the path of steepest ascent/descent. It was found for a spacing of $\Delta = 40$ that in no case were more than 10% of the total number of regions appropriate for first-order RSM. For $\Delta = 20$, the range of appropriate percentages varied from about 25% to 78%.

Again, this is a relevant, practical finding. Individuals conducting optimization must be very careful not to make experimental-region size too large, since then first-order parametric metamodels may only be appropriate 10% of the time, whereas setting even a smaller region size will still lead to considerable variability in achieving a properly executed search.

There are three implications of these findings. The first is that there is a need to develop a simulation-optimization "pre-processor" or "starter" that suggests both a starting point for the optimization and the granularity of the problem, i.e., the inter-gridpoint spacing or some surrogate. Many times it is appropriate to assume that a "good-enough" starting point is known by an expert, but even if so, it is not as clear that such an expert would have sufficient knowledge to specify an inter-gridpoint spacing that is not too big, given the particular model variabilities (exogenous and endogenous) and design conditions (run length and replications). Too small a spacing may be costly.

The second implication of the findings of this research is that nonparametric metamodeling should be examined. This is necessary not only because of the potential of violating parametric assumptions, but also for another reason implied in this research: the benefits of global, nonparametric metamodeling. Recall that in Figure 3.5 with two replications, multi-modal response surfaces were indicated (which was incorrect). If RSM were attempted starting on the wrong "side" of such a simulation response surface, the wrong optimum might be found. A possible alternative to parametric metamodeling such as RSM is global nonparametric metamodeling, whereby the whole surface is modeled using a nonparametric technique such as kernel smoothing or spline smoothing. In fact, some preliminary investigation on our part (Keys, Rees, Greenwood, 1995a) suggests that global, nonparametric metamodeling is very effective, seems safer, and requires relatively few computer runs to obtain the optimum.

The third implication of this research is that a multi-strategy approach to simulation optimization be explored. Since a response surface may vary considerably over the entire region in terms of both point and region characteristics/measures, it stands to reason that different search techniques might be appropriate and thus more successful in different areas of the search space. For example, RSM might be appropriate in one area and random search in another. We have initiated some discussion of this elsewhere (Crouch, Greenwood, Rees, 1995) (Greenwood, Rees, Crouch, 1993).

In conclusion, we recommend that further research be conducted to "flesh out" these three implications.

Chapter Four: A Best-First Search Approach for Determining Starting Regions In Simulation Optimization

INTRODUCTION

Definition of Simulation Optimization

Simulation is a widely-used computer modeling technique that has been applied to a broad scope of problems, ranging from traffic-flow analysis to job-shop scheduling to military-campaign planning. Simulation permits the study of systems which cannot feasibly be constructed or experimented upon in the "real world," and which are too complex to be analytically modeled. When a given set of input conditions is applied to a simulation model, the model's output, referred to as a response, provides an estimate of how the true system would respond to those inputs. Although simulation is very useful in predicting the output of a system or responses, it does not in and of itself indicate the input conditions required to achieve a desired response; i.e., it is not an optimization technique, it is an evaluative methodology. The

process of finding the input conditions that yield the optimal (or near optimal) system response(s) is referred to as simulation optimization, which can be a very expensive and time consuming activity. In other words, simulation evaluations address “what if” questions by providing performance measures for a given set of input conditions, whereas simulation optimization extends the evaluations to consider “what’s best” by seeking optimum values for the input conditions.

The objective of simulation optimization is to determine the values of the input conditions, n controllable factors or decision variables, that optimize m responses, subject to a set of uncontrollable conditions (conditions that affect outcomes but are not under the influence of the decision maker). This process is complicated by the presence of random error, often the result of combined random effects of all of the uncontrollable conditions. This causes a response Y_j to become a random variable and take on a set of values for the same setting of the controllable factors; i.e., there is some distribution of Y_j values for each combined level of the controllable factors. To model this behavior each response is oftentimes considered equal to the sum of a constant and a noise term, where the constant is the expected value of the response $E[Y_j]$ for a specific combination of factor settings, and the noise term represents the random error. Due to the presence of random error, the optimization process typically focuses on the expected value of the responses; however, while the goal is to optimize $E[Y_j]$, only Y_j is observable. Jacobson and Schruben (1989) note simulation optimization is in the class of stochastic optimization problems where the objective functions are stochastic functions of deterministic decision variables; these problems are known to be difficult to solve.

Azadivar (1992) points out that although the most common goal in simulation optimization is to optimize expected value, the goal may also involve such considerations as minimizing the risk of exceeding a threshold, minimizing dispersion, etc. Meketon (1987) refers to two classes of objectives of optimization procedures: min/max and level crossing (or root finding). The latter is of the form: find $X \ni E[Y(X)] = p$; for example, find the service rate such that customers wait more than 3 minutes 5% of the time. Meketon

also indicates that the level-crossing problem is the same as the min/max problem, e.g., $\min E[(Y(X)-p)^2]$, if $\text{Var}[Y(X)]$ is constant.

In general, the responses, $\mathbf{Y} = (Y_1, Y_2, \dots, Y_m)$, are functions of the controllable factors, $\mathbf{X} = (X_1, X_2, \dots, X_n)$, uncontrollable conditions, \mathbf{Z} , and random error, ε ; i.e.,

$$\mathbf{Y} = E[\mathbf{Y}] + \varepsilon = f(\mathbf{X} | \mathbf{Z}) = E[f(\mathbf{X} | \mathbf{Z})] + \varepsilon.$$

Note that the additive error considered above is only one possible model, with $E[\varepsilon_i] = 0$, and $\text{Var}[\varepsilon_i] < +\infty$.

In addition to the above goal, the optimization will be subject to upper and lower limits on the controllable factors or some function of a combination of them. Therefore, the general simulation optimization problem may be stated as:

$$\text{Optimize} \quad E[\mathbf{Y}] = E[f(\mathbf{X} | \mathbf{Z})] \text{ over the region } S \subset \mathfrak{R}^n \quad (1)$$

where the domain of S may be either continuous (\mathfrak{R}_c), or discrete (\mathfrak{R}_d), or mixed,

and $\mathbf{X} = (X_1, X_2, \dots, X_n) \in S$

Subject to:

$$h(\mathbf{X}) \geq 0 \quad (2)$$

where $\mathbf{h}(\mathbf{X})$ is a vector of deterministic constraints typically of the form:

$$l_i < X_i < u_i \quad i = 1, \dots, n \quad (2a)$$

$$l_{n+q} < f(\mathbf{X}) < u_{n+q} \quad q = 1, \dots, b \quad (2b)$$

where b is the number of constraints involving more than one controllable factor.

Typical Assumptions

Not all simulation optimization methods search the region S directly. For example, frequency domain methods transform the optimization problem into the frequency domain (Safizadeh, 1990), and many so-called *intrusive* procedures are single-simulation-run optimization methods (Wilson, 1987). However, a broad set of simulation optimization methods do explicitly perform a search directly over the region S . For example, different varieties of Response Surface Methodology (RSM, see Box and Wilson (1951) or Myers (1971)) assume a starting point in S then use first-order and/or second-order metamodels to suggest preferred directions of search or optimality locations. The research described in this chapter is most applicable to simulation optimization methods that search the region S directly, such as RSM, random search, and Box's complex search (Safizadeh, 1990), although any optimization approach that benefits from a carefully chosen initial region and/or requires a specification of problem granularity (see below) is a candidate for the procedures defined in this research.

Methods directly searching a region S typically make several assumptions. These often include the assumption that either a "good" starting point is known or that the choice of a starting point is unimportant to the solution of the problem. Sometimes this difficulty is obviated by selecting several starting points, solving the problem for each starting point, and selecting the most-preferred answer. Another assumption commonly invoked is that problem granularity, i.e., an appropriate grid spacing/step size, is known. For example, in using first-order RSM models, a factorial design is often utilized to determine the direction of steepest ascent. But there is no *a priori* rationale to determine the coding of natural variables in S , i.e., to specify the size of region over which the factorial design is defined. Moreover, once a direction of steepest ascent is determined from the RSM metamodel, there again is no *a priori* reasoning that leads to a good choice of step size along the path of steepest ascent. Finally, most approaches to simulation optimization invoke only one search method throughout the entire procedure, although some have suggested hybrid approaches (Crouch, Greenwood, and Rees 1995). Sometimes a

basic method is employed (e.g., RSM) with variations (e.g., first-order, second-order) to successfully address simulation models with differing amounts of curvature and/or variance in the response surface.

To summarize so far, many simulation optimization methods assume that a “good” starting point is identified, that the design grid (i.e., how far apart to space runs) is known, and that one basic search method need be employed, all regardless of the surface. Often, such assumptions are valid, for often a user has experience with the simulation model or is willing to live with the results obtained from assumptions, or expertise may be available to suggest appropriate search methods, step sizes, etc., early in the optimization process. Also, the surface may be “simple” and “smooth enough” to be impervious to the consequences of the aforementioned assumptions. However, there are cases where the simulation response surfaces are complex and have great variability in response across the surface and where little relevant optimization expertise is available. Ignoring these conditions can lead to an unnecessary expenditure of simulation runs, failure to find the simulation optimum, and/or a false declaration of the optimal conditions. Sometimes financial implications are significant. This research deals with this latter class of problems where making these assumptions is not wise.

The objective of this stream of research is to specify a “Starter” which can suggest a good starting point, a reasonable grid spacing, and an appropriate initial search methodology for those cases where these items are unknown and important. The Starter algorithm would be used initially in simulation optimization problems, and then would be followed by a conventional optimization method (such as RSM, Box’s complex search, etc.) using the starting point and inter-grid spacing stipulated by the Starter. In particular, this chapter locates a starting point and an inter-grid spacing. The objective of finding an initial appropriate search methodology is simplified in this chapter to finding a starting point and inter-grid spacing for a first-order RSM design. The Starter algorithm will be easy to modify to include other search techniques once the preferred conditions for starting with other search techniques is determined.

This latter matter is, of course, still an open research question (see, for example, (Crouch, Greenwood, and Rees 1995)).

Organizationally, this chapter is developed as follows. The next section provides a simulation optimization example that will be used throughout the chapter to illustrate concepts. This is followed by a discussion of the artificial-intelligence based search method, best-first search. The chapter then continues with the objectives and then implementation details of a Starter, and it is followed with three example simulation optimizations begun with the Starter. Finally, conclusions are drawn and further research directions are discussed.

ILLUSTRATIVE EXAMPLE

The concepts presented in this chapter are demonstrated through a primary example of an inventory simulation system with two variations. A second example, defined later, tests the procedure on a multi-modal surface.

The primary example is a simple inventory model that permits backorders, as illustrated in Figure 4.1. The model contains two decision variables or controllable factors -- order quantity (Q) and re-order point (R). Whenever the inventory level dips below the re-order point, an order of size Q is placed. The two controllable factors are varied during the search in order to find the combination of Q and R that yield the lowest total cost (TC). Total cost is composed of three components: ordering cost, carrying or holding cost, and shortage or backorder cost.

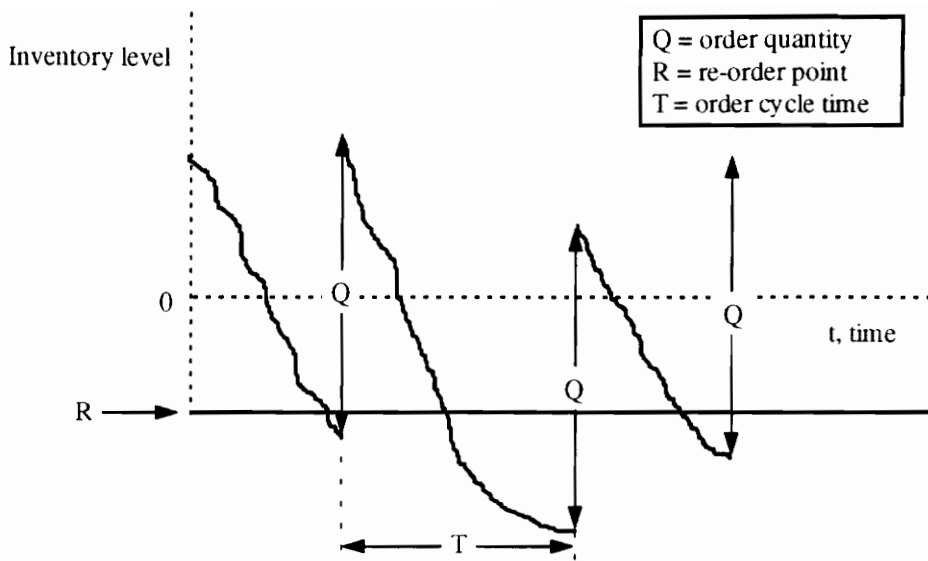


Figure 4.1. Simple inventory model that permits backorders and exhibits both stochastic demand and lead time.

The primary example model also contains two uncontrollable conditions. The first, interarrival times for demand (D), is a random variable which indirectly causes the inventory level to decrease at a non-constant rate, as illustrated in Figure 4.1. The second uncontrollable condition involves another random variable, lead time (L), the time between order placement and receipt. The effect of the stochastic lead time is that the inventory level does not always return to the same maximum value when an order of size Q is received, as is also shown in Figure 4.1.

In the primary example model, the random variable D is assumed to follow a Gamma(α , β) distribution, where the mean of D is $\alpha\beta$ and its variance is $\alpha\beta^2$. In the first variation of the primary example, lead time L is assumed to be identically zero. In the second variation, lead time is introduced and is assumed to follow a Normal distribution; in this second case, the mean of D remains the same as in the first case, but the variance of D is increased in order to illustrate the effect of high variability on the Starter strategy.

variables or controllable factors -- order quantity and re-order point, Q and R, respectively -- and random demand and lead time values that occurred during the simulated operation of the system. Every possible combination of Q and R, i.e., every point in (Q, R) space, represents a possible simulation run. In order to improve upon the expected total cost of the system, one changes the values of the decision variables and simulates the operation of the system again at another (Q, R) location. Decisions on how to change the value of the decision variables in order to get an improved solution occur in the "optimizer" box in Figure 4.2. The optimizer may involve a simple random strategy or a more complex but rational approach such as response surface methodology. Multiple simulation runs, replications, may be made at a single (Q, R) point in order to obtain a better estimate of the response (total cost) and to obtain an estimate of the variability of the response.

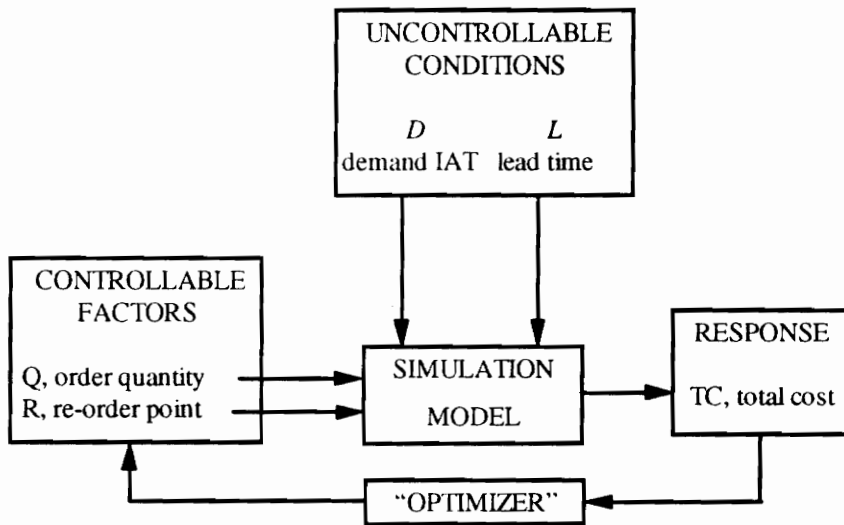


Figure 4.2. Process for optimizing the simulated inventory system

BEST-FIRST SEARCH

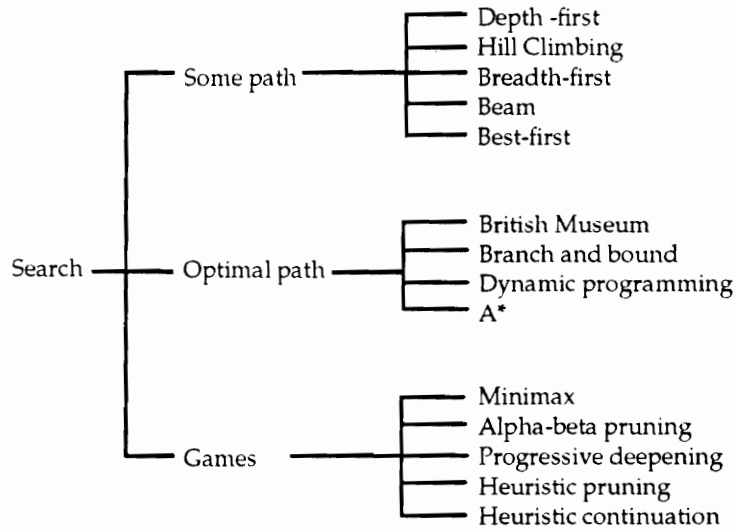
Best-First Search (BFS) is an AI-based search procedure that the *Encyclopedia of Artificial Intelligence* (1987) attributes to the work of Doran and Michie in 1966. This work presented an algorithm as part of a graph traversal program. It is this algorithm that has served as the basis for later variations of the BFS. We will explain and utilize BFS in a graph-traversal or network context as well. That is, we will invoke BFS as a network search whereby nodes represent subregions of the feasible area given by equation (2), and queues represent different paths through the network or search tree, as will be explained later.

The BFS procedure is heuristic in that it does not guarantee an optimal solution, but rather belongs to the class of search procedures that proceeds toward *some* solution, provided a solution exists (see Figure 4.3, based on (Winston, 1984)). BFS differs from some other heuristic AI search procedures in that estimates of the “goodness” of partial solutions are used to decide which further solutions to pursue. In many search situations, such estimates are unavailable, in which case BFS cannot be utilized; in those cases search strategies such as depth-first or breadth-first search must be invoked, where paths through a search tree are methodically developed regardless of the goodness of solutions discovered. In simulation optimization, measures are available as goodness estimates, such as the mean response at a point.

Pseudo code for the Best-First Search Algorithm as it proceeds from a Start node to a Goal node is shown in Figure 4.4; this code is based on (Winston, 1984).

A short example will illustrate the algorithm. Consider Figure 4.5a, which is a pictorial representation (a “map”) showing the connectivity of five cities. City a is the starting city, and we wish to travel to city b (our “goal”). Cities A, B, and C are intermediate cities, any or all of which we may travel through if we wish on our way from a to b . In Figure 4.5a the numbers on the arcs between cities represent actual distances between the cities. Our objective is to go from a to b in the shortest possible distance. Available

to us also is the estimated distance from each city to the goal, which in this example is an approximation based on the straight-line distance to the goal.

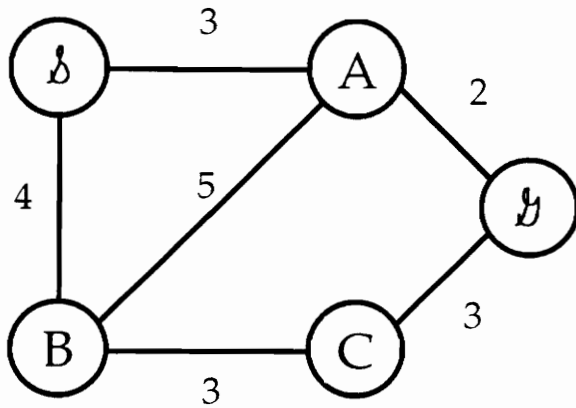


Note: Based on Winston [14]

Figure 4.3. Some AI-based search techniques

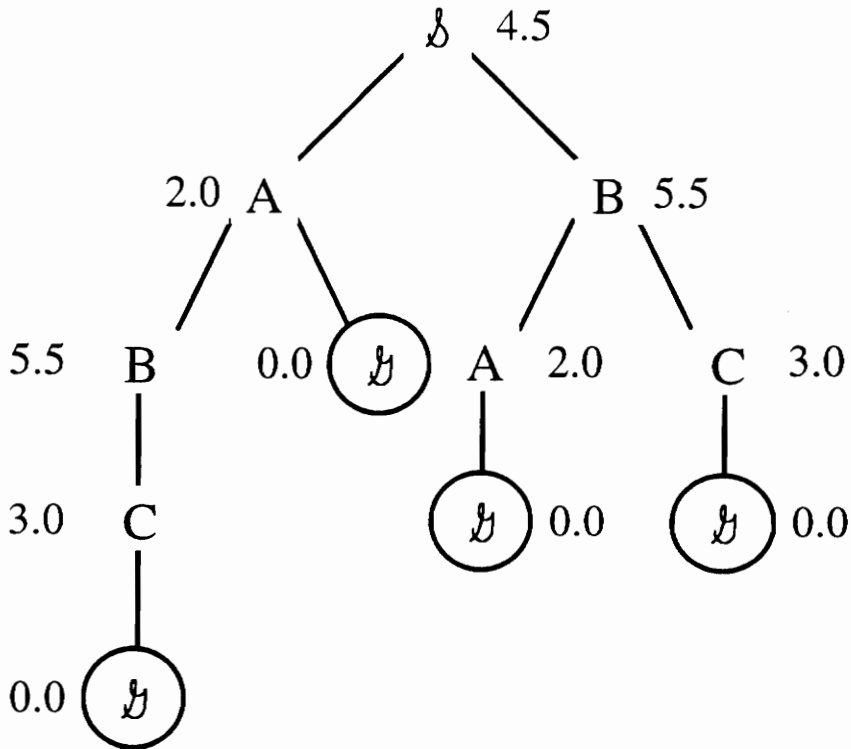
- Create a queue.
- Put the Start node on it.
- Until the Goal node has been reached or the queue is empty, repeat:
 - if the first element on the queue is NOT the goal node
 - remove the first element from the queue
 - add the first element’s children, if any, to the queue
 - sort the entire queue by estimated remaining distance to the goal
 - if the first element on the queue is the goal node
 - announce “success” and then STOP.
- If get to this step, the queue is empty and no Goal node has been found;
 - ∴ announce failure.

Figure 4.4. Best-First Search Algorithm psuedocode



Straight-line distance to goal	
From s	4.5
A	2.0
B	5.5
C	3.0
t	0.0

4.5a. A network representation of five cities



4.5b. The search tree corresponding to Figure 4.5a.

Figure 4.5. An example to illustrate the best-first search procedure

The search tree corresponding to the example is indicated in Figure 4.5b. From d , one may progress to either A or B. If one travels from d to A, then possible subsequent destinations are B and b . If one went from d to A and then to B, we assume that one could only travel to C. This is because going to d would be returning to a city already visited, for which there is clearly a shorter path. The rest of the tree is developed similarly.

The BFS algorithm begins by placing the source node d on the queue. As this node is not the goal node, the children of d , namely A and B, are added to the queue and d is removed. The contents of the queue at this point are

Queue

d -A : 2.0

d -B : 5.5

Here d -A indicates the path from d to A, d -B the path from d to B, and the numbers indicate the estimated distance to the goal. For example, the number 5.5 indicates that, once one is at B, the estimated straight-line distance to the goal is 5.5. According to the algorithm, the queue is next sorted based on estimated distance to the goal, which in this case results in no change at all. The first element on the queue is then removed from the queue and its children are added to the queue. This results in

Queue

d -A-B : 5.5

d -A- b : 0.0

d -B : 5.5

When the queue is sorted, it becomes

Sorted Queue

$h-A-h : 0.0$

$h-B : 5.5$

$h-A-B : 5.5$

The first path is removed from the sorted queue, and since it leads to the goal node, the procedure terminates successfully. The BFS solution to this problem is $h-A-h$, which (coincidentally) is the optimal path. Even though the example is simple, it illustrates that the procedure is desirable because the entire queue is sorted, thereby keeping the most promising alternative at the front of the queue. This is as opposed to depth-first search and breadth-first search, which place children at the front and back of the queue, respectively, and omit the sort. The tradeoff is one of effectiveness of the search procedure as opposed to the computational expense for the sorting. Computationally, BFS is of order n [$O(n)$] at worst and $O(\log n)$ at best, where n is the number of nodes, whereas depth-first and breadth-first search are $O(n)$. Of course, BFS is viable only when estimates to the goal are available.

STARTING A SIMULATION OPTIMIZATION SEARCH

Objectives

In the simulation optimization methodologies considered here, one wants to begin the search process at a point near the optimum, to explore a region of appropriate size near that starting point, and to use a

search methodology appropriate to the response surface of the simulation model. Often, however, one is unaware of where to start, how large a region to use, and/or what methodology is appropriate. For example, in the inventory simulation model described above, the user may only be able to state with confidence that $0 < Q^* \leq 400$ and $-400 \leq R^* \leq 0$, where (Q^*, R^*) is the optimal solution to the problem. Furthermore, the user may have no feel for the granularity of the problem other than to say that there is no need to place experimental design points closer than every 25 units in either the Q or R direction. It may be that a granularity of 50 or 100 units could suffice, but the user has no idea what the proper value is and whether the value changes as one gets closer to the optimum. Finally, the user may not know enough about the response surface itself to specify whether a gradient-based search method such as RSM will be appropriate, or whether another approach such as simulated annealing is more appropriate because the surface may be multimodal.

As a first step in building a simulation optimization starter, we assume the following objectives; our Starter should:

1. specify a “good” starting point (or points) for the subsequent search, in the sense that the optimal solution is likely to be found if the search is begun there.
2. suggest a minimum necessary inter-gridpoint spacing Δ_{\min} between experimental design points to be used initially in the subsequent search.
3. specify a subset of the global domain where the optimal solution(s) is (are) likely to reside; i.e., stipulate a limited search region.
4. generate another subset of the global domain where the optimal solution is almost certain *not* to reside.
5. allow the user to specify *aggressiveness*, e.g., to stipulate whether “many” runs should be expended to ensure good results (a not very aggressive strategy) or whether only a paucity of points should be investigated because, say, simulation runs are expensive or time consuming (a more aggressive strategy).
6. expend as few computer runs as possible within the context of the above objectives.

Note that we have omitted several important possible Starter objectives in this research; e.g., our Starter neither specifies a preferred search method, nor does it indicate which portions of the domain will be difficult to search (e.g., because of sharp peaks, multiple optima, ridges, etc.). These objectives are left as topics for future research, although we have initiated work on preferred search methods in (Crouch, Greenwood, and Rees 1995) and on global nonparametric metamodeling in (Keys, Rees, and Greenwood, 1995a; 1995b).

Objectives (2) and (6) taken together suggest a “divide-and-conquer” strategy for determining Δ_{\min} . For example, consider the simple domain $\mathcal{R}_1 = \{(x_1, x_2) \mid 0 \leq x_1 \leq 10, 0 \leq x_2 \leq 20\}$ shown in Figure 4.6. We initially make simulation runs at the corners of \mathcal{R}_1 , indicated by the four heavy dots in Figure 4.6a. To determine whether the (entire) region as specified stipulates a good value for Δ_{\min} in each of the x_1 and x_2 dimensions, we perform certain tests (given below) over the region. If the tests are passed, then there is no need to reduce Δ_{\min} , and $\Delta_{\min}^{(1)}$ in the x_1 direction is set to 10 and $\Delta_{\min}^{(2)}$ in the x_2 direction is set to 20. If, however, the test is not passed, then the Δ_{\min} s are too large. When this occurs, each dimension (i.e., x_1 and x_2) is bisected and additional runs are made as indicated in Figure 4.6b at the points (5,0), (5,10), (5,20), (0,10), and (10,10). This divides the region \mathcal{R}_1 into four new regions

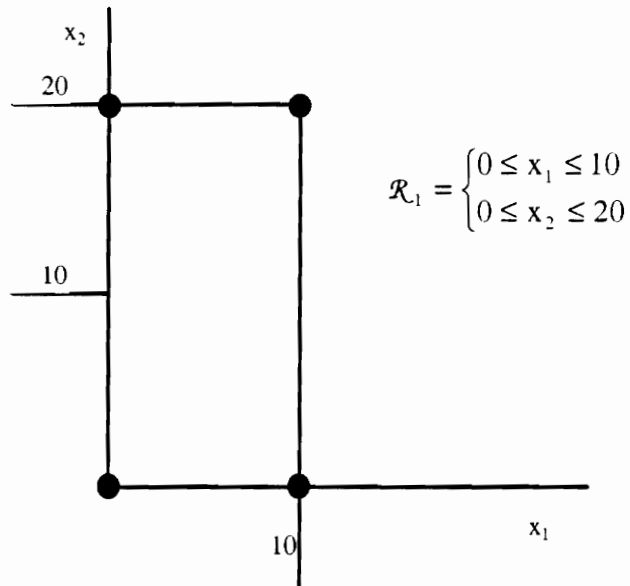
$$\mathcal{R}_{11} = \{(x_1, x_2) \mid 0 \leq x_1 \leq 5, 10 \leq x_2 \leq 20\},$$

$$\mathcal{R}_{12} = \{(x_1, x_2) \mid 5 \leq x_1 \leq 10, 10 \leq x_2 \leq 20\},$$

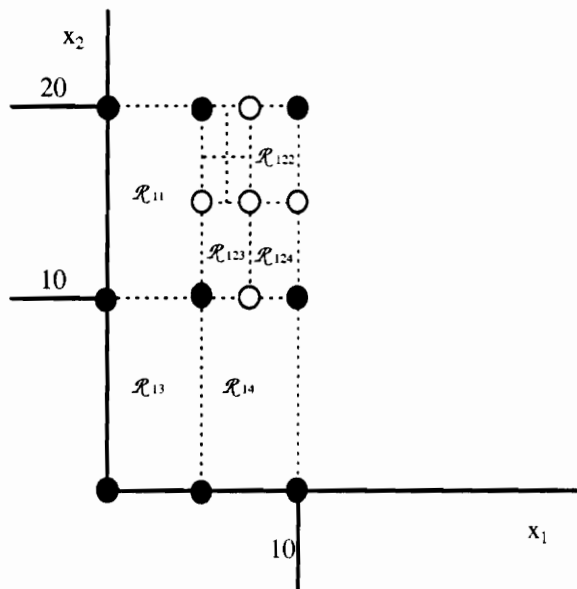
$$\mathcal{R}_{13} = \{(x_1, x_2) \mid 0 \leq x_1 \leq 5, 0 \leq x_2 \leq 10\},$$

$$\mathcal{R}_{14} = \{(x_1, x_2) \mid 5 \leq x_1 \leq 10, 0 \leq x_2 \leq 10\},$$

and implies $\Delta_{\min}^{(1)} = 5$ and $\Delta_{\min}^{(2)} = 10$. Again the tests are performed to see if the region is small enough to capture the essentials of the surface. If passed, the Δ_{\min} s are 5 and 10 respectively, and if not, the most preferred (again see below for how the preference is determined) region(s) is(are) quartered. For



4.6a. The domain of the simulation optimization problem with runs at the corners



4.6b. Additional runs made in the divide stage.

Figure 4.6. Using a divide-and-conquer strategy to determine design point spacing

example, if $\mathcal{R}_{12} \gg \mathcal{R}_{11}$ and $\mathcal{R}_{12} \gg \mathcal{R}_{13}$ and $\mathcal{R}_{12} \gg \mathcal{R}_{14}$ (where “ \gg ” means “is preferred”), then \mathcal{R}_{12} would be quartered (see the open circles in Figure 4.6b) to give

$$\mathcal{R}_{121} = \{(x_1, x_2) \mid 5 \leq x_1 \leq 7.5, 15 \leq x_2 \leq 20\},$$

$$\mathcal{R}_{122} = \{(x_1, x_2) \mid 7.5 \leq x_1 \leq 10, 15 \leq x_2 \leq 20\},$$

$$\mathcal{R}_{123} = \{(x_1, x_2) \mid 5 \leq x_1 \leq 7.5, 10 \leq x_2 \leq 15\},$$

$$\mathcal{R}_{124} = \{(x_1, x_2) \mid 7.5 \leq x_1 \leq 10, 10 \leq x_2 \leq 15\}.$$

This process is continued until the region tests are passed. For instance, region \mathcal{R}_{121} might be quartered itself, giving regions \mathcal{R}_{1211} , \mathcal{R}_{1212} , \mathcal{R}_{1213} , and \mathcal{R}_{1214} with corresponding minimal spacings of $\Delta_{\min}^{(1)} = 1.25$ and $\Delta_{\min}^{(2)} = 2.50$. Note that the divide-and-conquer strategy provides an estimate of Δ in a preferred region of the domain using relatively few simulation runs.

Whereas Starter objectives (2) and (6) imply a divide-and-conquer strategy, objectives (1) and (2) taken together imply a depth-first based search. To see this, consider Figure 4.7, which indicates a portion of the search tree used in finding the most preferred region (i.e., which region to start in and the corresponding region size). Since it is desired (objective (2)) to find the largest satisfactory Δ_{\min} in each direction, it is necessary to proceed down the tree as fast as possible until the region size tests are passed. At this point we note that searches that proceed down the search tree rather than across it are depth-first based searches. Since objective (1) asks that a “good” or most preferred starting point be found, the list of candidate regions to be searched should be searched from most preferred to least preferred, and explored in that order. But a depth-first search with the queue sorted from most preferred to least preferred at each step is equivalent to best-first search, as explained above. The BFS solution to the hypothetical problem above is indicated in Figure 4.7.

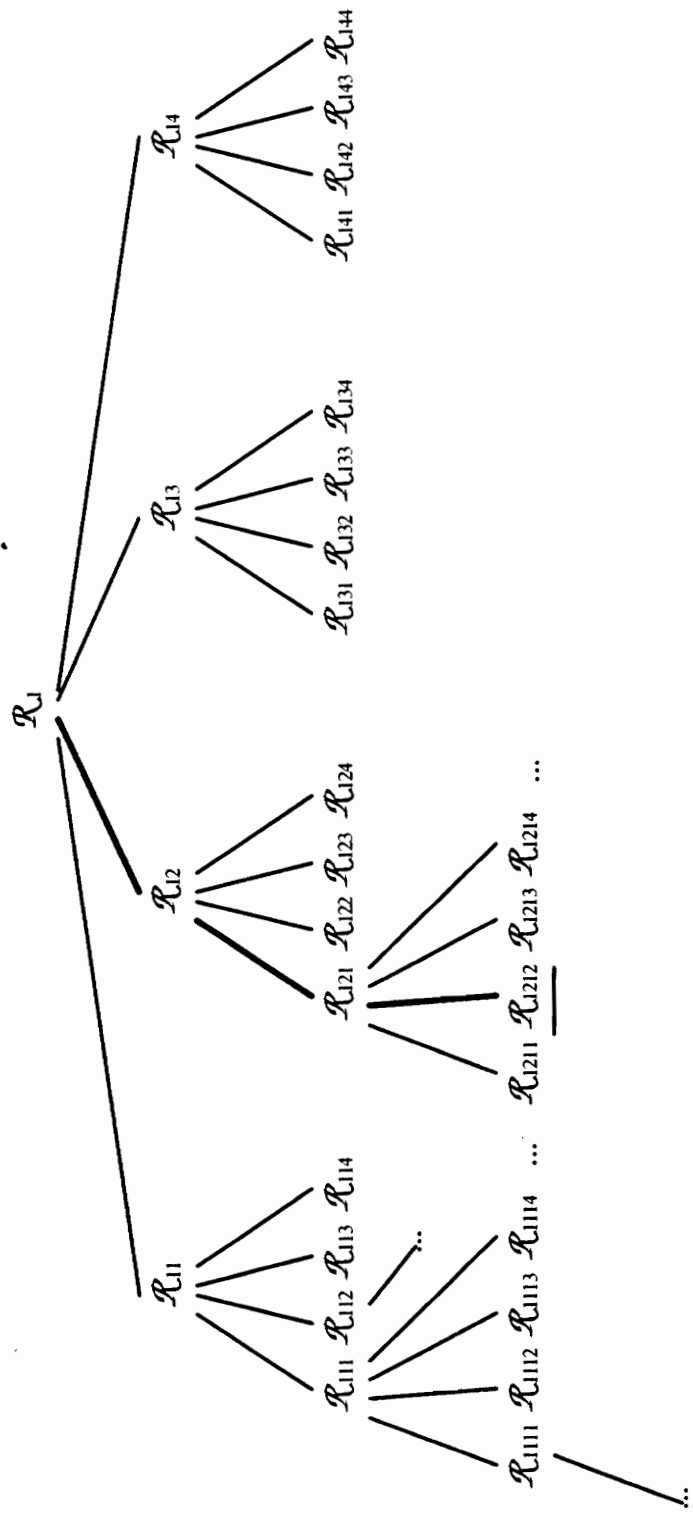


Figure 4.7. A portion of the search tree for finding good starting regions.

If a test can be devised to differentiate preferred from non-preferred regions, then objective (3) can be satisfied with a list of most promising regions. Similarly, such a test can also be used to identify a list of regions meeting objective (4), that subset of the domain almost certain not to include a good starting point.

Note that aggressiveness (objective (5)) or risk taking can be specified in several ways. First, a user may decide to use statistical tests and set a very low p-value for tests, thereby expressing conservatism. Similarly, the degree of aggressiveness may be set by specifying how regions may be eliminated from further consideration. For example, we specify below that all regions significantly different in a statistical sense from the best region discovered so far should be placed on a discard list. A less aggressive procedure might be to discard only those regions that are not different from any other region not different from the most preferred region. For example, if region \mathcal{R}_1 , the best region discovered so far, is not significantly different from region \mathcal{R}_2 , but region \mathcal{R}_3 is significantly worse than \mathcal{R}_1 but is not significantly worse than \mathcal{R}_2 , our Starter would place \mathcal{R}_3 on the discard list, whereas the less aggressive approach described above would not discard any of the three regions.

Having stated the objectives of our starter and their implications, we turn our attention in the next section to the identification of preferred regions and the implementation of the best-first search. Examples of the complete procedure are presented following this discussion.

Implementation of the Starter

In order to specify completely the BFS procedure for the Starter, we must stipulate the performance measure on which items in the queue will be sorted. Before doing this, we recall that BFS is a heuristic search and does not guarantee optimality.

Several different possible measures exist that make sense as indicators of region preference. For example, $\bar{Y}(\mathcal{R})$, the average of the response values obtained from the simulation runs made over the region, could be used. Alternatively, $\hat{Y}_{\max}(\mathcal{R}|M)$, the estimated maximum response over the region \mathcal{R} based on a first- or second-order metamodel M fitted to the simulated data, could be specified. This latter choice has the advantage of allowing goodness-of-fit tests as a means of helping to decide whether the Δ_{\min} s are small enough. But this estimate requires more simulation runs, particularly for a second-order metamodel, many of which may not be helpful in the initial stages of the search. Recall that the purpose of this procedure is to start the process of identifying promising regions to explore and does not necessarily have as an objective finding the system optimum. Because we want to demonstrate the concept with a relatively simple case, we choose $\bar{Y}(\mathcal{R})$ as our performance measure and leave open the question of other measures, such as $\hat{Y}_{\max}(\mathcal{R}|M)$, for future research.

Note that at this point we are choosing an aggressive strategy, or at least a strategy more aggressive than $\hat{Y}_{\max}(\mathcal{R}|M)$. Ultimately, the degree of aggressiveness could be set by the system at each stage of the search by specifying the most appropriate region preference measure. This topic is also left for future research.

With $\bar{Y}(\mathcal{R})$ as the performance measure and a divide-and-conquer approach using best-first search, the Starter strategy becomes clear. The essential features of the Starter are shown in Figure 4.8.

In the algorithm of Figure 4.8 the subdividing is done by bisecting the most preferred region along each dimension, as explained. Note also that the entire queue (list) \mathcal{E} is sorted from the most-preferred to the least-preferred region, as is required in best-first search. However, two items must be further explained. First, it has not yet been shown how statistical tests are performed that determine which regions should be

placed on the list \mathcal{E} and which on the discard list \mathcal{D} . Second, the stopping criteria have also not been stipulated for the Starter.

User: Initialize
 The user specifies the dimensionality (k) of the simulation-optimization problem.
 The user specifies the region to be optimized; call it \mathcal{R}_1 .
 The user specifies whether optimization is minimization or maximization.

System: Define and Initialize
 Initially define \mathcal{D} =list of regions to be discarded= $\{\emptyset\}$.
 Define each vertex of the region \mathcal{R}_1 as a "gridpoint."
 Run replications, e.g., three, at each gridpoint.
 Initially define \mathcal{E} =list of promising regions to be explored= $\{\mathcal{R}_1\}$.
 Define Δ_i as the inter-gridpoint spacing between gridpoints along dimension i , $i = 1, \dots, k$.

Best-first search
 Repeat
 While the list \mathcal{E} is not empty
 Take the first region \mathcal{R}_i off \mathcal{E} .
 While not meeting stop criteria
 Subdivide \mathcal{R}_i .
 Perform statistical tests on all regions \mathcal{R}_j on \mathcal{E} .
 Based on these tests, keep promising regions on \mathcal{E} and place others on \mathcal{D} .
 Sort \mathcal{E} , putting the most promising region at the front of the list.
 End While /* not meeting stop criteria */
 End While /* the list \mathcal{E} is not empty */
 Run the Safety Net
 Until the lists \mathcal{E} and \mathcal{D} are empty.

Figure 4.8. The basic Starter algorithm.

The testing is a multiple-comparison test of all regions on the list \mathcal{E} of regions still to be explored. With this test, the most preferred region and all those not significantly different from it are kept on the list \mathcal{E} , whereas all regions significantly different from the most preferred region are placed on the discard list.

The proper multiple-comparison test depends on whether heteroskedasticity is present; the particular procedure used in our Starter follows Toothaker (1991) and is as follows:

- For each region \mathcal{R} taken one at a time on the list \mathcal{E} , assume a metamodel M and determine ϵ_j from $y_j = M + \epsilon_j$, where j represents a run at a gridpoint in the region \mathcal{R} .
- Use the Shapiro-Wilk test to test for normality on all ϵ_j for each region.
- Assume homoskedasticity unless disproved below.
- If any region has nonnormal errors then
 - if Levene's Median test is significant
 - then heteroskedasticity is present;
 - else
 - if the Bartlett-Box test is significant
 - then heteroskedasticity is present.
- If homoskedasticity is present then
 - use the Tukey-Kramer multiple comparison procedure;
 - else
 - use Scheffe's multiple comparison procedure.

As mentioned, the metamodel $M = \bar{Y}(\mathcal{R})$ is used in this research, but other metamodels should be explored in future research.

There are currently three rules that serve as the stopping criteria in the best-first search. Terminology and notation used in the stopping criteria are

F_{REGR} = the F-value of a significance-of-regression test, and

F_{LOF} = the F-value of a lack-of-fit test.

The rules are

1. Stop if the user-supplied minimum inter-gridpoint spacing (call it Δ_u) is reached.
2. Stop if gradient-based search methods, e.g. RSM, can be used properly in the most-preferred region \mathcal{R}^* . I.e., stop if F_{REGR} is significant and F_{LOF} is not significant for \mathcal{R}^* .
3. Stop if a horizontal hyperplane may be fit accurately over \mathcal{R}^* . This implies that any point in the most-preferred region is optimal and there is no further reason to divide this region. I.e., stop if F_{REGR} and F_{LOF} both are not significant for \mathcal{R}^* .

Because the penalty for incorrectly placing a region on the discard list is very high, namely an optimal solution may be discarded, we invoke a safety-net in our strategy. This provides a “second chance” to place regions on the discard list back under consideration. The safety-net is invoked after the list \mathcal{E} is emptied; comparison of each item on the list \mathcal{D} is made with the best region discovered so far (\mathcal{R}^*). No region is permanently discarded that is larger than $4\Delta_{\text{min}}$ without subdividing that region first. Details of our implementation of the safety-net as well as pseudo code for the entire Starter are included in Appendix B. The examples that follow will also include detail not specified here, but included in Appendix B.

EXAMPLES

In this section three examples are presented in varying levels of detail to illustrate the Starter procedure. The first example is the inventory model with low variance demand and no lead time described above. The second example considers the same model but with more variance in the demand and stochastic lead time. The third example consists of a multi-modal response surface with five peaks and an annular

depression around the highest peak. The purpose of the third example is to test the starter on a complex, multi-modal surface where gradient-based searches could fail.

Example 1: Inventory Model with Low-Variance Demand and No Lead Time.

Further simulation model specification. We assume in this example that the simulation model used by the user has the following parameters:

$$L = \text{lead time} = 0$$

$$D = \text{daily demand interarrival time} \sim \Gamma(\alpha, \beta),$$

where $\alpha=1.0$, $\beta=0.2$, which is also $\exp(0.2)$. Note that with an assumption of 250 days per year and a demand inter-arrival time of 0.2 days, the expected number of arrivals per year is $250 \times 5 = 1250$.

Simulation run length ≥ 4 years of 250 days each; i.e., the simulation ends at the completion of the first cycle at or beyond 1000 days. (See Appendix A for further details on “cycles.”) Warm-up period ≥ 250 days; i.e., the warm-up period ends at the completion of the first cycle at or beyond 250 days.

$$C_h = \text{holding cost} = \$10/\text{unit}/\text{year}$$

$$C_b = \text{backorder cost} = \$5/\text{unit}/\text{year}$$

$$C_o = \text{ordering cost} = \$50/\text{order}$$

User specification. The user stipulates that the dimensionality of the problem is two ($k=2$), and that the decision variables are (Q, R) . Furthermore, the objective function is the minimization of daily total cost. The optimal values to this simulation model are believed to lie in the following region, as far as the user knows:

$$16 \leq Q \leq 400$$

$$-384 \leq R \leq 0$$

$$|R| \leq Q \leq -16.$$

(We have allowed the user to choose a lower bound on Q of 16 and a lower bound on R of -384 for ease of exposition. By choosing these values, the bisection of Q and R results in nicer numbers for the boundaries of regions.) Note that the user's specification of the region yields a triangular shape, thereby implying that the Starter will generate not only rectangular subregions as above, but triangular subregions as well.

Finally, the user stipulates that spacing gridpoints 24 units apart in either the Q or the R direction will provide sufficient granularity, i.e., $\Delta_u^{(1)} = \Delta_u^{(2)} = \Delta_u = 24$.

“True” answer to this problem. The expected optimal solution can be found by solving the closed-form (analytic), deterministic model of this problem, allowing only integer values for Q and R . However, it is assumed that the “true” response surface generated by this simulation model is unknown to the user. The optimal solution is

$$Q = 194$$

$$R = -130.$$

$$TC = \$2.58 \text{ per day,}$$

and a contour plot of the true response surface is indicated in Figure 4.9.

Notation. The BFS Starter solution to this example is shown in the various panels of Figure 4.10; in these panels we show the progress of the solution as well as the final answer. To explain the solution, three items of notation must be explained. We define the list \mathcal{E} as the list of regions to be explored further and the list \mathcal{D} as the list of regions to be discarded (for later analysis on the safety-net). Both lists

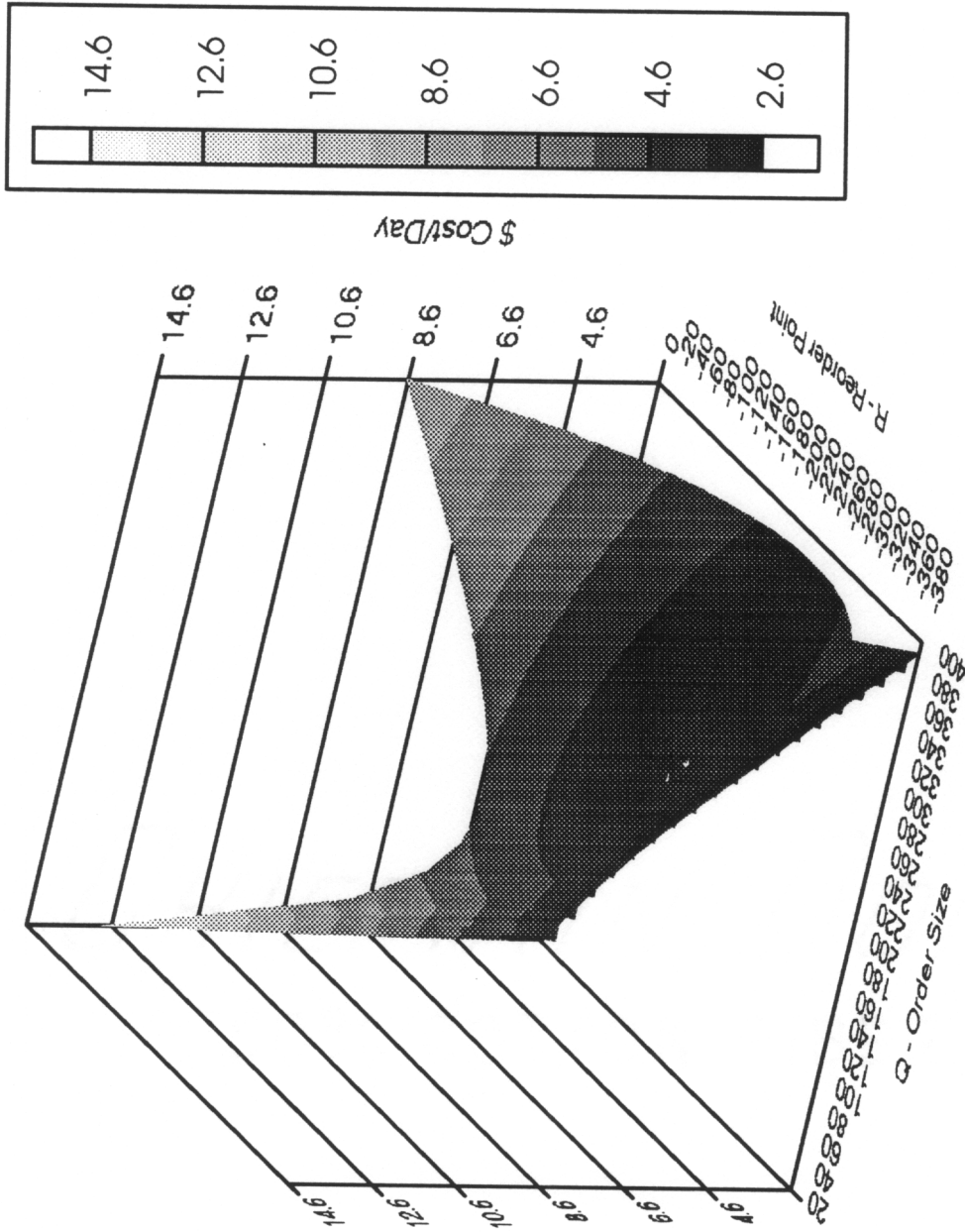


Figure 4.9. A contour plot of the theoretical response surface for Example 1

initially are empty, i.e., $\mathcal{E} = \{\phi\}$ and $\mathcal{D} = \{\phi\}$. In addition we use superbars to indicate regions not statistically different from each other. For example, the notation

$$\left\{ \overline{\bar{Y}_{212}}, \overline{\bar{Y}_{341}}, \overline{\bar{Y}_2}, \overline{\bar{Y}_{22}}, \overline{\bar{Y}_{14}}, \bar{Y}_5, \bar{Y}_{213} \right\}$$

indicates that the average response in regions \mathcal{R}_{212} and \mathcal{R}_{341} do not differ statistically from each other; moreover, neither do regions \mathcal{R}_2 , \mathcal{R}_{22} , and \mathcal{R}_{14} differ. The absence of a superbar indicates that each of the regions \mathcal{R}_5 and \mathcal{R}_{213} does differ from all the other regions on the list.

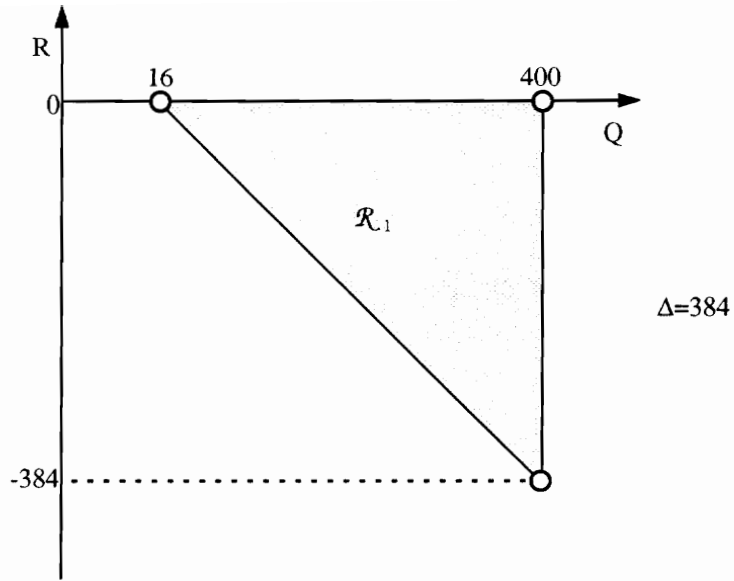
Initialization. The procedure begins with three replications of simulation runs at each of the three vertices of the user-defined feasible region (call it \mathcal{R}_1), as indicated in Figure 4.10a by the open circles. Note that the initial inter-gridpoint spacing is $\Delta^{(1)}_{\min} = \Delta^{(2)}_{\min} = \Delta_u = 384$. The list \mathcal{E} becomes $\mathcal{E} = \{\mathcal{R}_1\}$, while the list $\mathcal{D} = \{\phi\}$.

Best-first search. (1). The first (and only) region on \mathcal{E} , namely \mathcal{R}_1 , is removed from \mathcal{E} and is divided by bisecting \mathcal{R}_1 along each dimension, thereby generating three new regions \mathcal{R}_{11} , \mathcal{R}_{12} , and \mathcal{R}_{13} (see Figure 4.10b); these regions are defined as

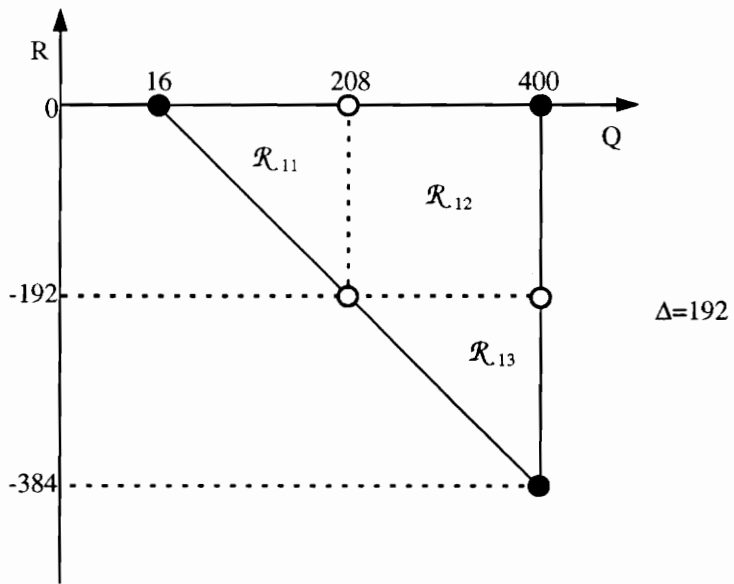
$$\mathcal{R}_{11}: 16 \leq Q \leq 208; -192 \leq R \leq 0; |R| \leq Q - 16;$$

$$\mathcal{R}_{12}: 208 \leq Q \leq 400; -192 \leq R \leq 0;$$

$$\mathcal{R}_{13}: 208 \leq Q \leq 400; -384 \leq R \leq -192; |R| \leq Q - 16.$$



4.10a. The initial (user-specified) search region \mathcal{R}_1



4.10b. The first pass through the best-first search loop

(cont.)

Note that regions \mathcal{R}_{11} and \mathcal{R}_{13} are triangular, whereas region \mathcal{R}_{12} is rectangular (square). Inter-gridpoint spacing is now $\Delta_{\min}^{(1)} = \Delta_{\min}^{(2)} = \Delta_{\min} = 192$. The three regions formerly comprising \mathcal{R}_1 replace it on \mathcal{E} , giving $\mathcal{E} = \{\mathcal{R}_{11}, \mathcal{R}_{12}, \mathcal{R}_{13}\}$. \mathcal{D} remains = $\{\phi\}$. The best region located so far is $\mathcal{R}^* = \mathcal{R}_1$.

Simulation runs (three replications) are made at those corner points of the new regions where runs have not already been made, i.e., at the open circles in Figure 4.10b. All runs both old and new in each region (a total of 9 in \mathcal{R}_{11} , 12 in \mathcal{R}_{12} , and 9 in \mathcal{R}_{13}) are averaged to provide the estimates \bar{Y}_{11} , \bar{Y}_{12} , and \bar{Y}_{13} . The first multiple comparison test is now made. It compares the means of the following regions, in general: the union of (a) all regions newly subdivided at this step, and (b) \mathcal{R}^* , if and only if \mathcal{R}^* is not the region that was subdivided at this step. The idea is to compare the costs of the newly formed regions with the cost of the best region discovered so far, in order to immediately prune inferior regions from the list of regions to explore. In this case, $\mathcal{R}^* = \mathcal{R}_1$ was subdivided, so the multiple comparison is made of only the subdivided regions \mathcal{R}_{11} , \mathcal{R}_{12} , and \mathcal{R}_{13} . In order to see whether the means of these three regions differ statistically from each other, the following procedure is followed:

- The Shapiro-Wilk test is used to check for normality for each region \mathcal{R}_i on \mathcal{E} using the model $y_{ij} = \bar{Y}_i + \epsilon_{ij}$, where j represents the j th simulation run in the i th region. (Note that if there are g gridpoints in the i th region, each with r replications, then $n_i = g*r$ simulation runs will be made in region i .) Results indicate that regions \mathcal{R}_{11} and \mathcal{R}_{12} have nonnormal errors, whereas there is no violation of the normality assumption in region \mathcal{R}_{13} .

- The appropriate test for homogeneity of variance over $\mathcal{R}_1 = (\mathcal{R}_{11} \cup \mathcal{R}_{12} \cup \mathcal{R}_{13})$ is Levene's Median test, since the normality assumption is violated over the region \mathcal{R}_1 . Results indicate that heteroskedasticity is present, thereby implying that Scheffe's test is the appropriate multiple comparison test.
- The result of Scheffe's multiple comparison test is

$$\overline{\overline{Y}}_{13} \quad \overline{\overline{Y}}_{12} \quad \overline{\overline{Y}}_{11}.$$

Note that the notation implies that the mean in region \mathcal{R}_{13} is preferred to that in \mathcal{R}_{12} , which in turn is preferred to the mean in \mathcal{R}_{11} . But the means in regions \mathcal{R}_{11} and \mathcal{R}_{12} do not differ statistically from each other, and the same may be said about \mathcal{R}_{12} and \mathcal{R}_{13} . Since the mean of \mathcal{R}_{11} does differ statistically from the most preferred (i.e., lowest cost) mean, \mathcal{R}_{11} is removed from \mathcal{E} and is placed on \mathcal{D} , leaving $\mathcal{E} = \{\mathcal{R}_{13}, \mathcal{R}_{12}\}$, and $\mathcal{D} = \{\mathcal{R}_{11}\}$. Note that the order of regions in our notational scheme is significant in that the list is sorted from most-preferred region to least-preferred. The best region located so far is $\mathcal{R}^* = \mathcal{R}_{13}$.

(2). The best-first search loop is repeated, with the first region on \mathcal{E} , namely \mathcal{R}_{13} removed. \mathcal{R}_{13} is subdivided, giving the three new subregions shown in Figure 4.10c and described by

$$\mathcal{R}_{131}: 208 \leq Q \leq 304; -288 \leq R \leq -192; |R| \leq Q - 16;$$

$$\mathcal{R}_{132}: 304 \leq Q \leq 400; -288 \leq R \leq -192;$$

$$\mathcal{R}_{133}: 304 \leq Q \leq 400; -384 \leq R \leq -288; |R| \leq Q - 16.$$

Note that inter-gridpoint spacing is now $\Delta_{\min}^{(1)} = \Delta_{\min}^{(2)} = \Delta_{\min} = 96$. Also note that Figure 4.10c only shows the old region \mathcal{R}_{13} and not the entire feasible region.

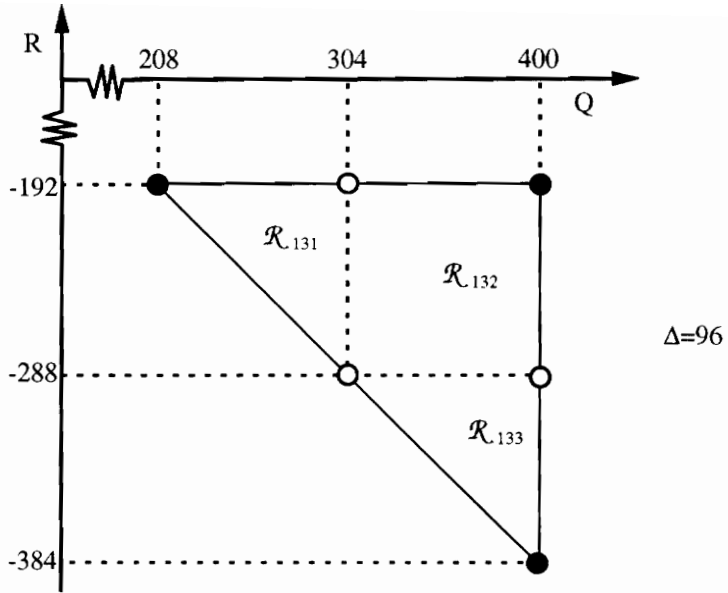
Simulation runs (three replications) are made at those corner points of the new regions where runs have not already been made, i.e., at the open circles in Figure 4.10c. Once again, R^* is subdivided, so only the newly divided regions are included in the multiple comparison test. Therefore, all runs in each of the three newest regions are averaged to provide the estimates \bar{Y}_{131} , \bar{Y}_{132} , and \bar{Y}_{133} .

- The Shapiro-Wilk test shows violation of the normality assumption.
- Levene's Median test indicates that heteroskedasticity is not present, thereby implying that the Tukey-Kramer test is the appropriate multiple comparison test.
- The result of that multiple comparison test is

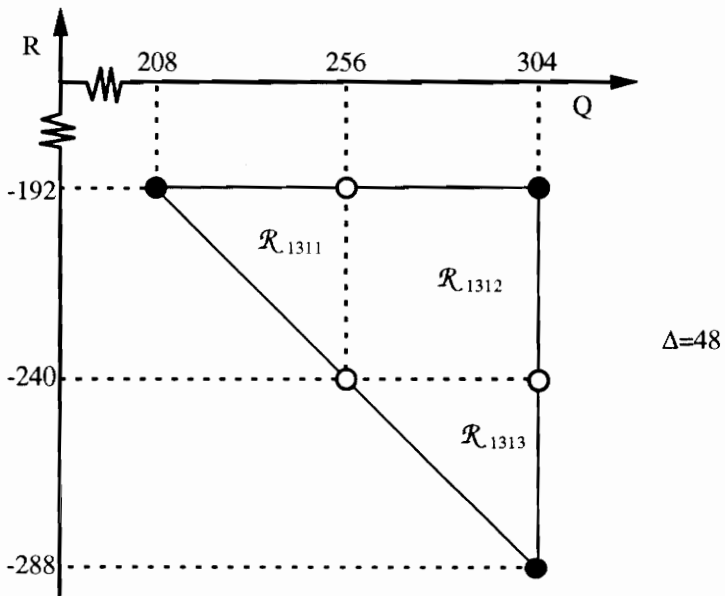
$$\overline{\overline{\overline{Y_{131}}}} \quad \overline{\overline{\overline{Y_{132}}}} \quad \overline{\overline{\overline{Y_{133}}}}$$

Therefore, \mathcal{R}_{133} is removed from \mathcal{E} and is placed on \mathcal{D} , leaving (after sorting in ascending cost order for each list) $\mathcal{E} = \{\mathcal{R}_{131}, \mathcal{R}_{132}, \mathcal{R}_{12}\}$, and $\mathcal{D} = \{\mathcal{R}_{133}, \mathcal{R}_{11}\}$.

Having determined which of the subdivided regions should be placed on \mathcal{E} and on \mathcal{D} , the multiple comparison test is now repeated for the entire list \mathcal{E} . This is done because there is now a new best (lowest



4.10c. The second pass through the best-first search loop



4.10d. The third pass through the best-first search loop

(cont.)

cost) region ($\mathcal{R}^* = \mathcal{R}_{131}$), so it may be possible to remove some of the higher cost regions at the end of the list \mathcal{E} . Shapiro-Wilk shows nonnormality, Levene's Median test shows heteroskedasticity, and Scheffe's test gives

$$\overline{\overline{Y_{131} \ Y_{132} \ Y_{12}}}$$

Hence, the \mathcal{E} and \mathcal{D} lists are unchanged.

(3). The best-first search loop is repeated again, with the first region on \mathcal{E} , namely \mathcal{R}_{131} removed. The region \mathcal{R}_{131} is subdivided, giving the three new subregions shown in Figure 4.10d and described by

$$\mathcal{R}_{1311}: 208 \leq Q \leq 256; -240 \leq R \leq -192; |R| \leq Q - 16;$$

$$\mathcal{R}_{1312}: 256 \leq Q \leq 304; -240 \leq R \leq -192;$$

$$\mathcal{R}_{1313}: 256 \leq Q \leq 304; -288 \leq R \leq -240; |R| \leq Q - 16.$$

Inter-gridpoint spacing is now $\Delta_{\min}^{(1)} = \Delta_{\min}^{(2)} = \Delta_{\min} = 48$.

Simulation runs (three replications) are made at those corner points of the new regions where runs have not already been made, i.e., at the open circles in Figure 4.10d. The multiple comparison test is of the three new regions, so all runs in each of these regions are averaged to provide the estimates \overline{Y}_{1311} , \overline{Y}_{1312} , \overline{Y}_{1313} .

- The Shapiro-Wilk test shows no violation of the normality assumption, implying that the Bartlett-Box test for homogeneity of variance is appropriate.

- The Bartlett-Box test indicates that heteroskedasticity is not present, thereby making the Tukey-Kramer test the appropriate multiple comparison test.
- The result of that multiple comparison test is

$$\overline{\overline{\overline{Y}_{1312} Y_{1311} Y_{1313}}}$$

Therefore, \mathcal{R}_{1312} and \mathcal{R}_{1311} are placed on \mathcal{E} , and \mathcal{R}_{1313} is placed on \mathcal{D} , leaving (after sorting) $\mathcal{E} = \{\mathcal{R}_{1312}, \mathcal{R}_{1311}, \mathcal{R}_{132}, \mathcal{R}_{12}\}$, and $\mathcal{D} = \{\mathcal{R}_{1313}, \mathcal{R}_{133}, \mathcal{R}_{11}\}$.

The multiple comparison test is now repeated for the entire list \mathcal{E} , because there is a new $\mathcal{R}^* = \mathcal{R}_{1312}$. Shapiro-Wilk shows nonnormality, Levene's Median test shows heteroskedasticity, and Scheffe's test gives

$$\overline{\overline{\overline{\overline{Y}_{1312} Y_{1311} Y_{132} Y_{12}}}}$$

Hence, \mathcal{R}_{12} is put on \mathcal{D} so $\mathcal{E} = \{\mathcal{R}_{1312}, \mathcal{R}_{1311}, \mathcal{R}_{132}\}$, and $\mathcal{D} = \{\mathcal{R}_{1313}, \mathcal{R}_{133}, \mathcal{R}_{12}, \mathcal{R}_{11}\}$.

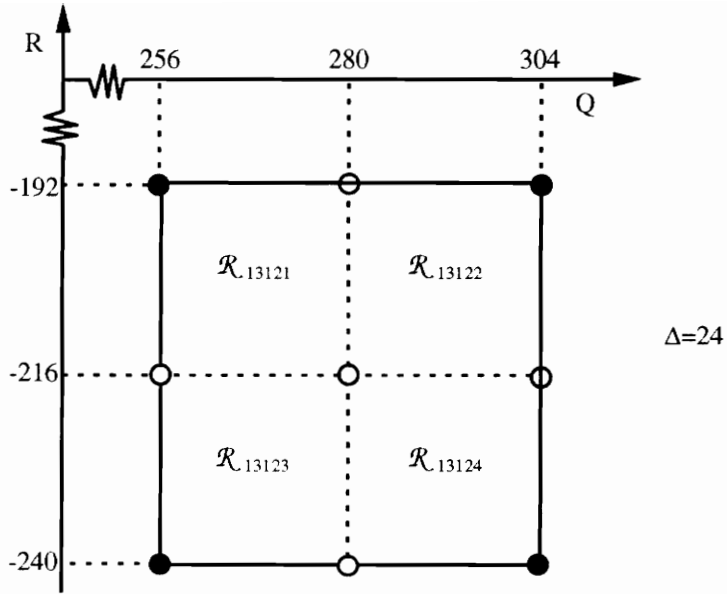
(4). The best-first search loop is repeated for the fourth time, with the first region on \mathcal{E} , namely \mathcal{R}_{1312} removed. \mathcal{R}_{1312} is subdivided, giving the four new subregions shown in Figure 4.10e and described by

$$\mathcal{R}_{13121}: 256 \leq Q \leq 280; -216 \leq R \leq -192;$$

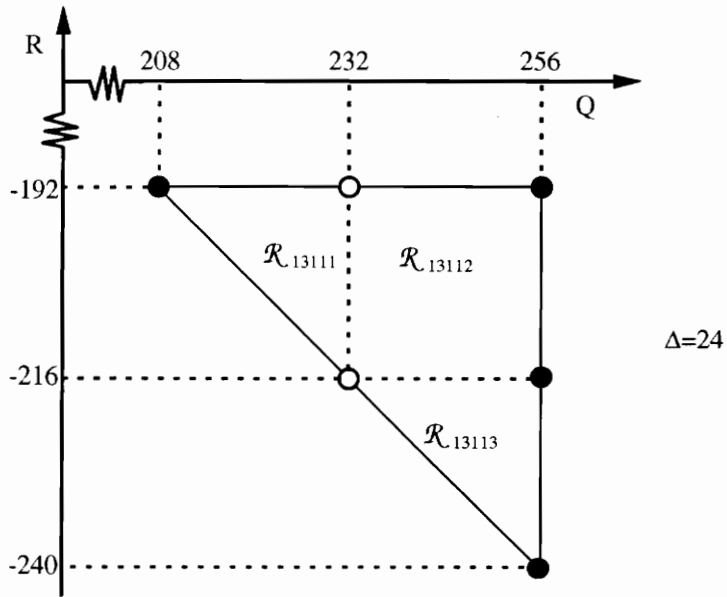
$$\mathcal{R}_{13122}: 280 \leq Q \leq 304; -216 \leq R \leq -192;$$

$$\mathcal{R}_{13123}: 256 \leq Q \leq 280; -240 \leq R \leq -216;$$

$$\mathcal{R}_{13124}: 280 \leq Q \leq 304; -240 \leq R \leq -216.$$



4.10e. The fourth pass through the best-first search loop



4.10f. The fifth pass through the best-first search loop

Figure 4.10. The BFS Starter solution to Example 1.

Inter-gridpoint spacing is now $\Delta_{\min}^{(1)} = \Delta_{\min}^{(2)} = \Delta_u = 24$, which is also the user-specified minimum inter-gridpoint spacing (Δ_u).

Simulation runs (three replications) are made at those corner points of the new regions where runs have not already been made, i.e., at the open circles in Figure 4.10e. Again the multiple comparison test is of the new regions, so averages are made of the runs in each of these four regions to provide the estimates

$$\bar{Y}_{13121}, \bar{Y}_{13122}, \bar{Y}_{13123}, \text{ and } \bar{Y}_{13124}.$$

- The Shapiro-Wilk test shows a violation of the normality assumption, implying that the Levene-Median test for homogeneity of variance is appropriate.
- The Levene-Median test indicates that heteroskedasticity is present, thereby making the Scheffe test the appropriate multiple comparison test.

The result of that multiple comparison test is

$$\overline{\overline{\bar{Y}_{13121}}} \overline{\overline{\bar{Y}_{13122}}} \overline{\overline{\bar{Y}_{13124}}} \overline{\overline{\bar{Y}_{13123}}}$$

Therefore, \mathcal{R}_{13121} and \mathcal{R}_{13122} are placed on \mathcal{E} , and \mathcal{R}_{13124} and \mathcal{R}_{13123} are placed on \mathcal{D} , leaving

(after sorting) $\mathcal{E} = \{\mathcal{R}_{13121}, \mathcal{R}_{13122}, \mathcal{R}_{1311}, \mathcal{R}_{132}\}$, and $\mathcal{D} = \{\mathcal{R}_{13124}, \mathcal{R}_{13123}, \mathcal{R}_{1313}, \mathcal{R}_{133}, \mathcal{R}_{12},$

$\mathcal{R}_{11}\}$. Since the best region located so far is a new region, $\mathcal{R}^* = \mathcal{R}_{13121}$, the multiple comparison test is

now repeated for the entire list \mathcal{E} . Shapiro-Wilk shows nonnormality, Levene's Median test shows heteroskedasticity, and Scheffe's test gives

$$\overline{\overline{\overline{\bar{Y}_{13121}}}} \overline{\overline{\overline{\bar{Y}_{13122}}}} \overline{\overline{\bar{Y}_{1311}}} \overline{\bar{Y}_{132}}.$$

Consequently, \mathcal{R}_{132} is placed on \mathcal{D} leaving $\mathcal{E} = \{\mathcal{R}_{13121}, \mathcal{R}_{13122}, \mathcal{R}_{1311}\}$, and $\mathcal{D} = \{\mathcal{R}_{13124}, \mathcal{R}_{13123}, \mathcal{R}_{1313}, \mathcal{R}_{132}, \mathcal{R}_{133}, \mathcal{R}_{12}, \mathcal{R}_{11}\}$.

At this point, \mathcal{R}_{13121} and \mathcal{R}_{13122} are removed from \mathcal{E} , because they are at the user-supplied inter-gridpoint minimum (i.e., 24). This leaves $\mathcal{E} = \{\mathcal{R}_{1311}\}$.

(5). The fifth loop through the best-first search takes \mathcal{R}_{1311} off of \mathcal{E} . \mathcal{R}_{1311} is subdivided, giving the three new subregions shown in Figure 4.10f and described by

$$\mathcal{R}_{13111}: 208 \leq Q \leq 232; -216 \leq R \leq -192; |R| \leq Q - 16;$$

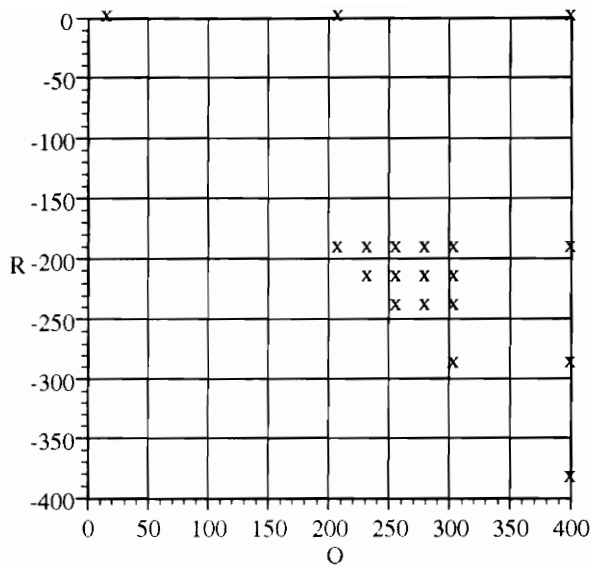
$$\mathcal{R}_{13112}: 232 \leq Q \leq 256; -216 \leq R \leq -192;$$

$$\mathcal{R}_{13113}: 232 \leq Q \leq 256; -240 \leq R \leq -216; |R| \leq Q - 16.$$

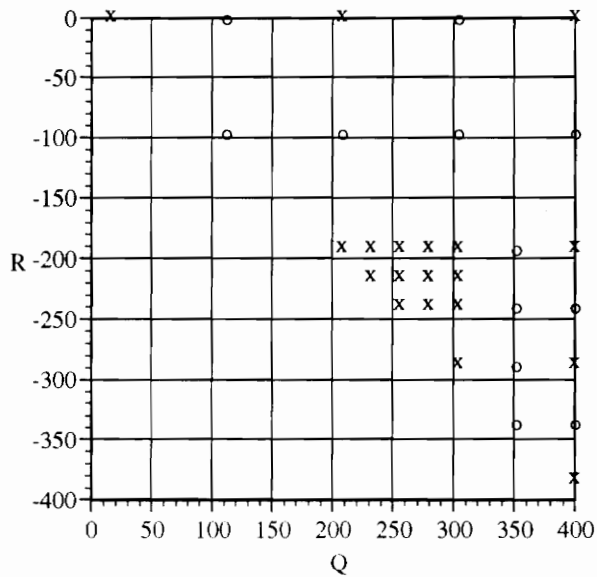
Inter-gridpoint spacing is now $\Delta_{\min}^{(1)} = \Delta_{\min}^{(2)} = \Delta_{\min} = 24$, which is also the user-specified minimum inter-gridpoint spacing (Δ_0).

Simulation runs (three replications) are made at those corner points of the new regions where runs have not already been made, i.e., at the open circles in Figure 4.10f. Since $\mathcal{R}^* = \mathcal{R}_{13121}$ is not one of the newly divided regions, the multiple comparison test is of \mathcal{R}^* and the three new regions. All runs in each region are averaged to provide the estimates $\bar{Y}_{13121}, \bar{Y}_{13111}, \bar{Y}_{13112}$, and \bar{Y}_{13113} .

- The Shapiro-Wilk test shows no violation of the normality assumption, implying that the Bartlett-Box test for homogeneity of variance is appropriate.



4.11a. Gridpoints where simulation runs are made (x) during the best-first search portion of the search.



4.11b. Gridpoints where simulation runs are made (o) during the first pass of the safety net.

Figure 4.11. The location of simulation runs in Example 1

Details of the safety-net procedure will not be elaborated here; rather, only an overview and summary results will be given. Each “pass” through the safety net examines every region \mathcal{R} on the list \mathcal{D} one at a time and compares each \mathcal{R} with the best region \mathcal{R}^* found so far either during the best-first search part of the algorithm or during a previous pass of the safety net. The comparison of each \mathcal{R} with \mathcal{R}^* results in either \mathcal{R} being discarded (i.e., removed from all further consideration) or placed back on the list \mathcal{E} of candidate regions. No region is discarded, without further testing, that is greater than or equal to four times the smallest inter-gridpoint spacing Δ encountered for any region thus far. A “pass” through the safety net ends whenever the list \mathcal{D} is depleted and all new items placed on \mathcal{E} from \mathcal{D} have been run through the best-first search portion of the algorithm. Of course, the repetition of the BFS part of the search may result in further items being placed on \mathcal{D} , which requires another “pass” through the safety net. The entire Starter algorithm terminates when there are no more regions on either \mathcal{E} or \mathcal{D} ; i.e., the Starter terminates when all regions have been discarded.

A very aggressive strategy with respect to the safety net is to conduct no passes through the safety net at all. Conversely, a very conservative (cautious) safety-net strategy is to conduct all necessary passes to clear \mathcal{E} and \mathcal{D} (i.e., run the algorithm to completion). A more moderate approach, and the strategy used here, is to perform one pass through the safety net. This gives every region \mathcal{R} in the list \mathcal{D} “one last chance” to compete with \mathcal{R}^* , the best region found so far.

In example 1, one complete pass through the safety net takes an additional 12 points (or 36 runs). The location of these 12 safety net points is shown with small circles in Figure 4.11b. The best starting point at the end of the first pass is then

Starting point: $Q = 184$
 $R = -120$
 $TC = \$2.59/\text{day}.$

Note that the total cost figure is very close to the optimal daily cost (within one cent per day). However, the best-first search procedure left several regions on \mathcal{D} , so the entire safety-net procedure may be invoked again if the user wishes to be very cautious. In fact to entirely empty the list \mathcal{D} , the safety-net must be invoked twice more for a total of 27 more points (and 81 runs); the same (Q, R) point and total cost is obtained as after the first pass of the safety net. If the entire BFS Starter procedure is run from start to finish with all passes of the safety net a total of 237 runs at 79 points is made. This is roughly half the 459 runs at 153 points needed if, rather than using the Starter, the entire region is blanketed with points at an inter-gridpoint spacing of 24.

Example 2: Inventory Example with Higher Demand Variance and Lead Time Variance

Further simulation model specification. In order to demonstrate the Starter procedure on a simulation model with more variability in response, the example above is modified to include lead time L and to increase the variance of the interarrival time for daily demand D . In particular,

$L \sim \text{truncated } N(6, 2^2)$, i.e., $L \geq 0$, rather than $L = 0$ in example 1, and

D is changed to $D \sim \Gamma(\alpha_D, \beta_D)$, where $\alpha_D = 0.05$, $\beta_D = 4.0$.

With these parameters, the mean of demand remains the same as before, but demand variance is increased by a factor of twenty. All other parameters and costs remain the same as in Example 1.

User specification. As before, the user stipulates that the dimensionality of the problem is two ($k=2$), the decision variables are (Q, R) , and the objective function is the minimization of daily total cost. The optimal values to this simulation model are believed to lie in the following region, as far as the user knows:

$$48 \leq Q \leq 400$$

$$-352 \leq R \leq 0$$

$$|R| \leq Q \leq -48.$$

(Once again the lower bounds on Q and R have been assumed set by the user to 48 and -352 respectively in order to make the numbers “nicer” for the example, and also because the introduction of lead time in this example restricts (Q, R) policies if one wishes to guarantee feasibility.)

For this example, we assume the user specifies no minimum inter-gridpoint spacing, Δ_u . As such, we use the system default, namely a Δ_u allowing for 5 repetitions of the “divide (i.e., bisect) and conquer” steps

which is $\left(\frac{1}{2}\right)^5 * 100\% = 3.125\%$ of the initial range. In this case $\Delta_u = 0.03125 * 352 = 11$ is permitted.

“True” answer to this problem. Figure 4.12a shows the expected simulation response surface, based on three replications, whereas, Figure 4.12b illustrates the variance of the response surface under the same conditions. The true optimal solution is:

$$Q = 194$$

$$R = -130.$$

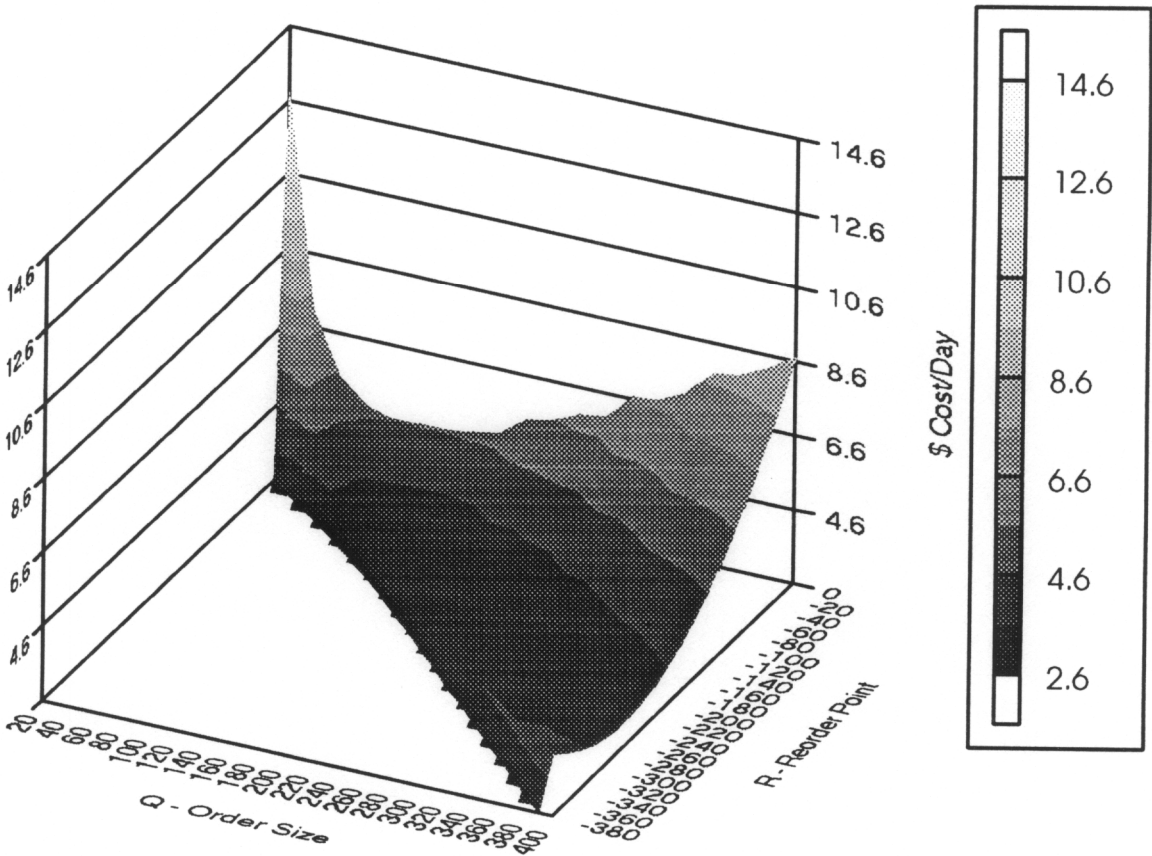
$$TC = \$2.58 \text{ per day.}$$

Initialization. The procedure begins with three replications of simulation runs at each of the three vertices of the user-defined feasible region (call it \mathcal{R}_1), as indicated in Figure 4.13a by the open circles. Note that

the initial inter-gridpoint spacing is $\Delta_{\min}^{(1)} = \Delta_{\min}^{(2)} = \Delta_{\min} = 352$. The list \mathcal{E} becomes $\mathcal{E} = \{\mathcal{R}_1\}$, while the list $\mathcal{D} = \{\phi\}$.

Best-first search. (1). The first (and only) region on \mathcal{E} , namely \mathcal{R}_1 , is removed from \mathcal{E} and is divided as shown in Figure 4.13b. The results of the best-first search steps are $\mathcal{E} = \{\mathcal{R}_{13}, \mathcal{R}_{11}, \mathcal{R}_{12}\}$, $\mathcal{D} = \{\phi\}$,

$\Delta_{\min}^{(1)} = \Delta_{\min}^{(2)} = \Delta_{\min} = 176$, and $\mathcal{R}^* = \mathcal{R}_{13}$.

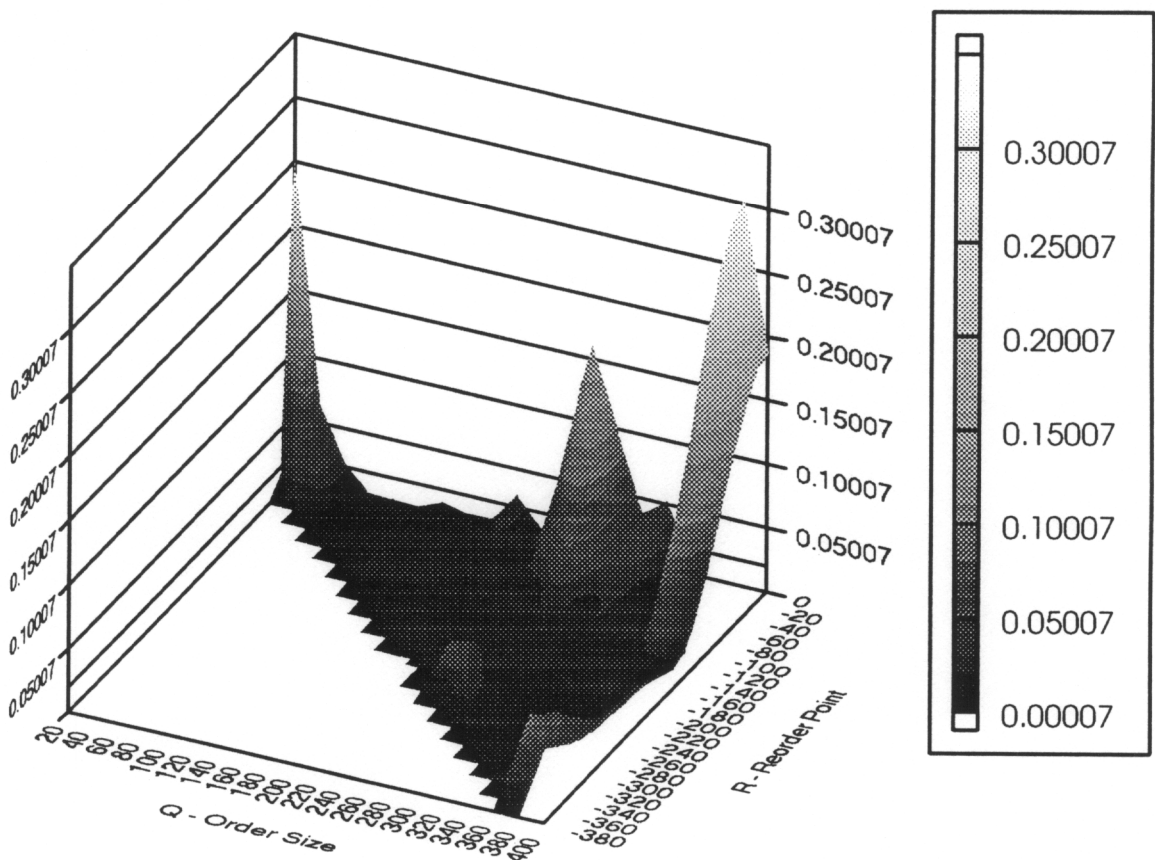


4.12a. The mean of the response

(cont.)

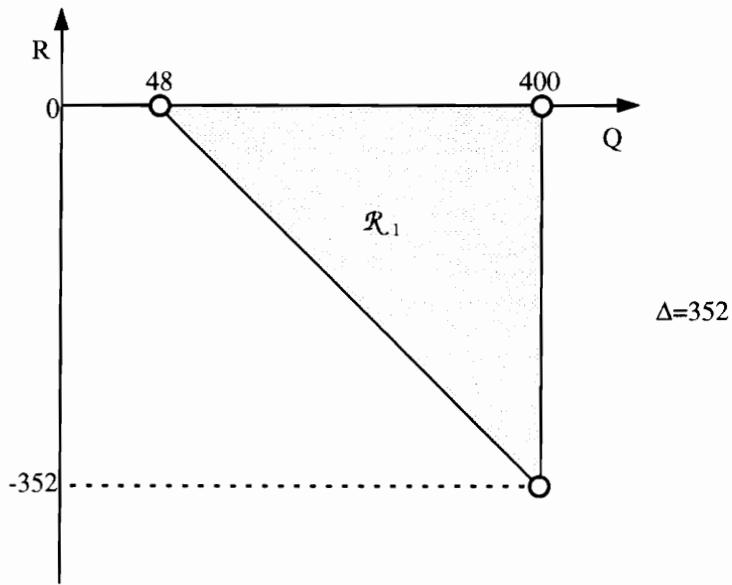
(2). The first region on \mathcal{E} , \mathcal{R}_{13} , is removed from \mathcal{E} and is divided as shown in Figure 4.13c. The results of the best-first search steps are $\mathcal{E} = \{\mathcal{R}_{131}, \mathcal{R}_{12}, \mathcal{R}_{11}\}$, $\mathcal{D} = \{\mathcal{R}_{132}, \mathcal{R}_{133}\}$, $\Delta_{\min}^{(1)} = \Delta_{\min}^{(2)} = \Delta_{\min} = 88$, and $\mathcal{R}^* = \mathcal{R}_{131}$.

(3). The first region on \mathcal{E} , \mathcal{R}_{131} , is removed from \mathcal{E} and is divided as shown in Figure 4.13d. The results of the best-first search steps are $\mathcal{E} = \{\mathcal{R}_{1311}, \mathcal{R}_{1312}, \mathcal{R}_{12}\}$, $\mathcal{D} = \{\mathcal{R}_{1313}, \mathcal{R}_{11}, \mathcal{R}_{132}, \mathcal{R}_{133}\}$, $\Delta_{\min}^{(1)} = \Delta_{\min}^{(2)} = \Delta_{\min} = 44$, and $\mathcal{R}^* = \mathcal{R}_{1311}$.

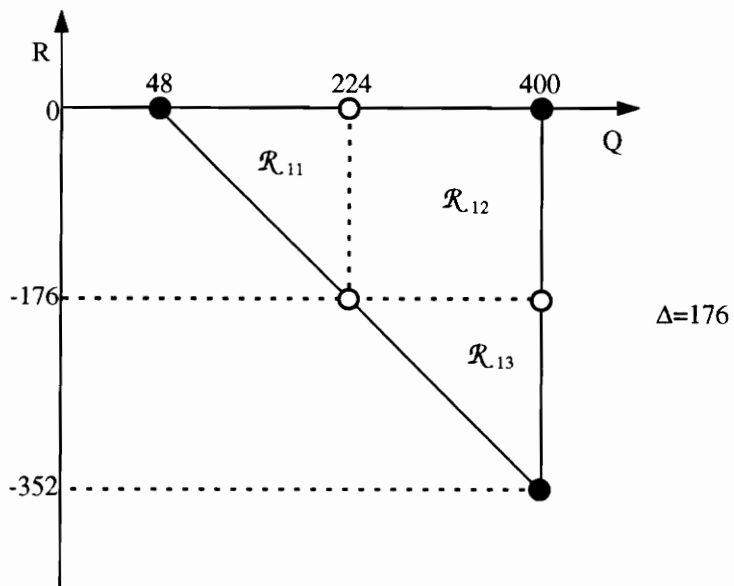


4.12b. The variance of the response

Figure 4.12 Some response surface characteristics for the model of Example 2

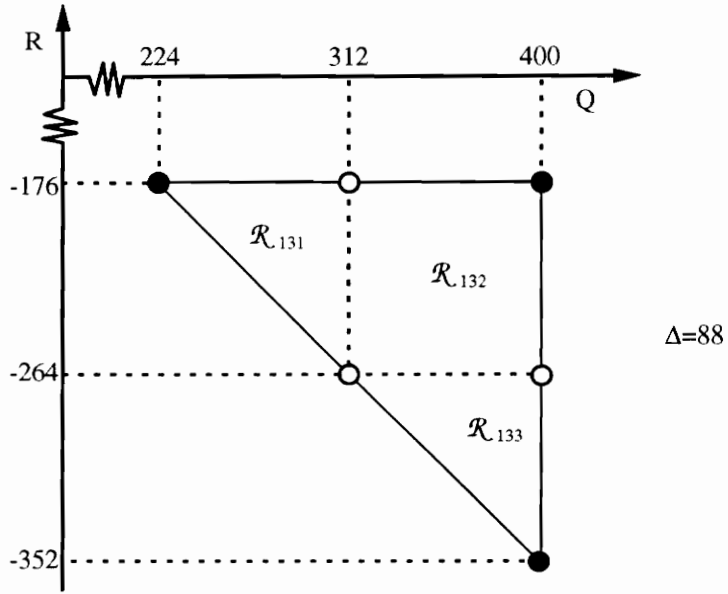


4.13a. The initial (user-specified) search region \mathcal{R}_1

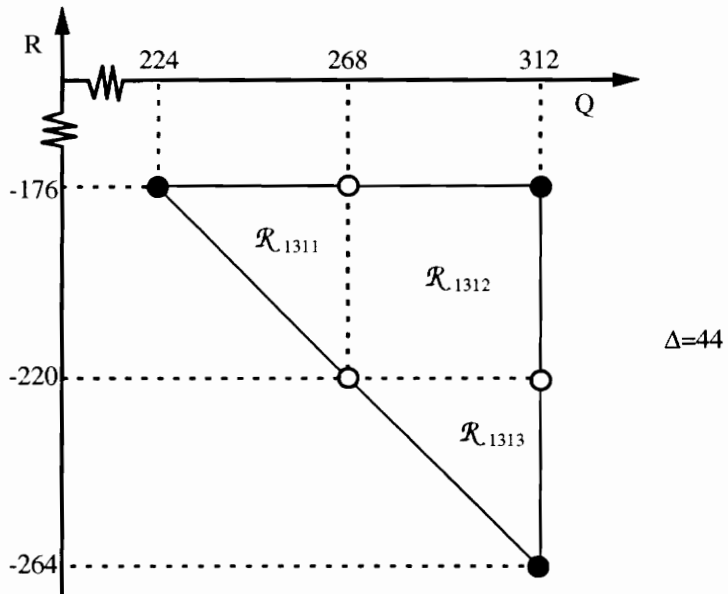


4.13b. The first pass through the best-first search loop

(cont.)



4.13c. The second pass through the best-first search loop



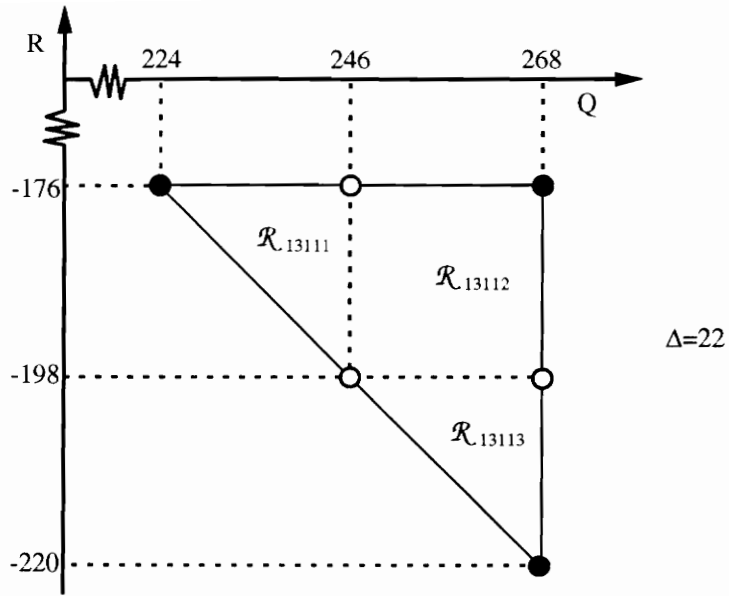
4.13d. The third pass through the best-first search loop

(cont.)

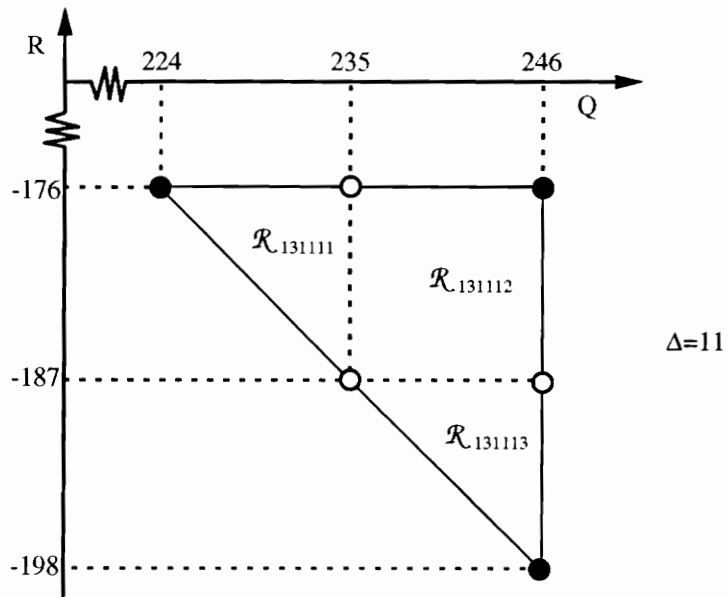
(4). The first region on \mathcal{E} , \mathcal{R}_{1311} , is removed from \mathcal{E} and is divided as shown in Figure 4.13e. The results of the best-first search steps are $\mathcal{E} = \{\mathcal{R}_{13112}, \mathcal{R}_{13111}, \mathcal{R}_{13113}, \mathcal{R}_{1312}, \mathcal{R}_{12}\}$, $\mathcal{D} = \{\mathcal{R}_{1313}, \mathcal{R}_{11}, \mathcal{R}_{132}, \mathcal{R}_{133}\}$, $\Delta_{\min}^{(1)} = \Delta_{\min}^{(2)} = \Delta_{\min} = 22$, and $\mathcal{R}^* = \mathcal{R}_{13112}$.

(5). The first region on \mathcal{E} , \mathcal{R}_{13112} , is removed from \mathcal{E} and is determined to meet the stopping condition specified in rule 3 above, i.e., the plane fit through region \mathcal{R}_{13112} is horizontal. As such, this region is removed from the list \mathcal{E} , giving $\mathcal{E} = \{\mathcal{R}_{13111}, \mathcal{R}_{13113}, \mathcal{R}_{1312}, \mathcal{R}_{12}\}$. Next, the new first element on \mathcal{E} , \mathcal{R}_{13111} , is removed and is divided as shown in Figure 4.13f. The results of the best-first search steps are $\mathcal{E} = \{\mathcal{R}_{131112}, \mathcal{R}_{131111}, \mathcal{R}_{131113}, \mathcal{R}_{13113}, \mathcal{R}_{1312}\}$, $\mathcal{D} = \{\mathcal{R}_{12}, \mathcal{R}_{1313}, \mathcal{R}_{11}, \mathcal{R}_{132}, \mathcal{R}_{133}\}$, $\Delta_{\min}^{(1)} = \Delta_{\min}^{(2)} = \Delta_{\min} = 11$, and $\mathcal{R}^* = \mathcal{R}_{13112}$.

(6). The first region on \mathcal{E} , \mathcal{R}_{131112} , is removed from \mathcal{E} and is also determined to meet the stopping condition specified in rule 2 above. As this region is “RSM-able,” it is marked as such and is removed from the list \mathcal{E} , giving $\mathcal{E} = \{\mathcal{R}_{131111}, \mathcal{R}_{131113}, \mathcal{R}_{13113}, \mathcal{R}_{1312}\}$. Next, the new first element on \mathcal{E} , \mathcal{R}_{131111} , is removed. But subdividing this region gives a $\Delta_{\min} = 5.5 < \Delta_u = 11$; therefore, \mathcal{R}_{131111} is removed from \mathcal{E} . The same is true for \mathcal{R}_{131113} . This leaves $\mathcal{E} = \{\mathcal{R}_{13113}, \mathcal{R}_{1312}\}$. Therefore, region \mathcal{R}_{13113} is removed from \mathcal{E} and subdivided as shown in Figure 4.13g. The results of the best-first search steps are $\mathcal{E} = \{\mathcal{R}_{131132}, \mathcal{R}_{131131}\}$, $\mathcal{D} = \{\mathcal{R}_{1312}, \mathcal{R}_{131133}, \mathcal{R}_{12}, \mathcal{R}_{1313}, \mathcal{R}_{11}, \mathcal{R}_{132}, \mathcal{R}_{133}\}$, $\Delta_{\min}^{(1)} = \Delta_{\min}^{(2)} = \Delta_{\min} = 11$, and $\mathcal{R}^* = \mathcal{R}_{13112}$.

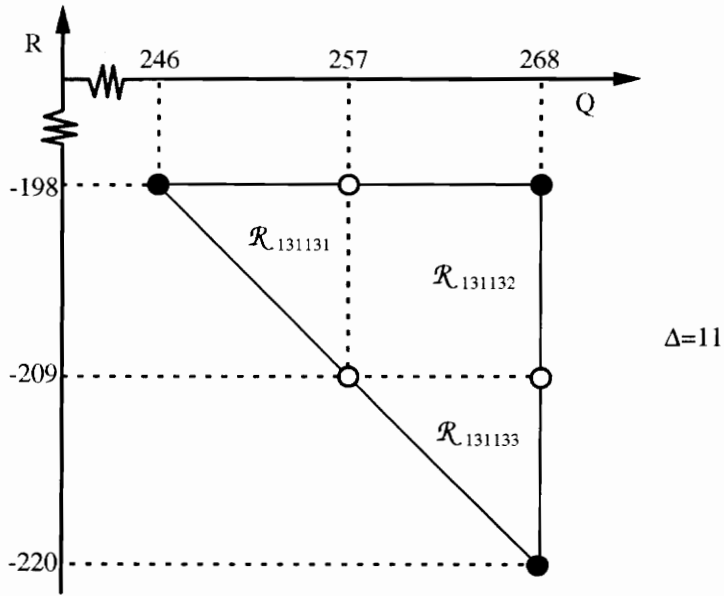


4.13e. The fourth pass through the best-first search loop



4.13f. The fifth pass through the best-first search loop

(cont.)



4.13g. The sixth pass through the best-first search loop

Figure 4.13. The BFS Starter solution to Example 2.

Since further dividing either region on \mathcal{E} results in $\Delta_{\min} < \Delta_u$, the list \mathcal{E} becomes empty, and the best-first search portion of the Starter ends.

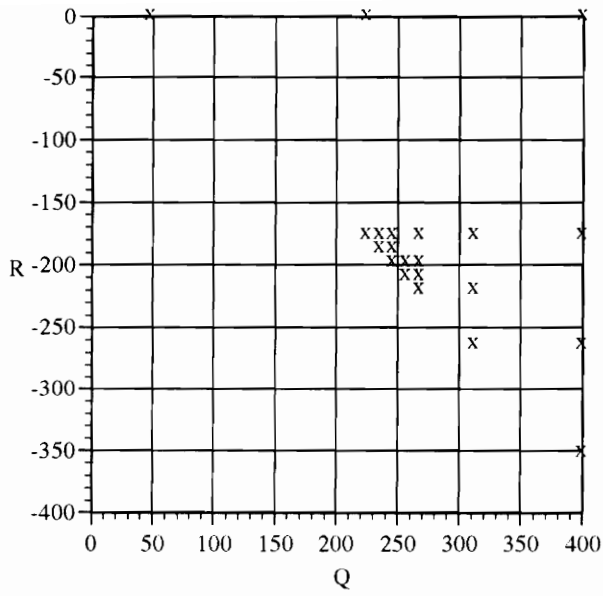
The results obtained from the Starter without the safety net are as follows; start search at

Starting point: $Q = 268$

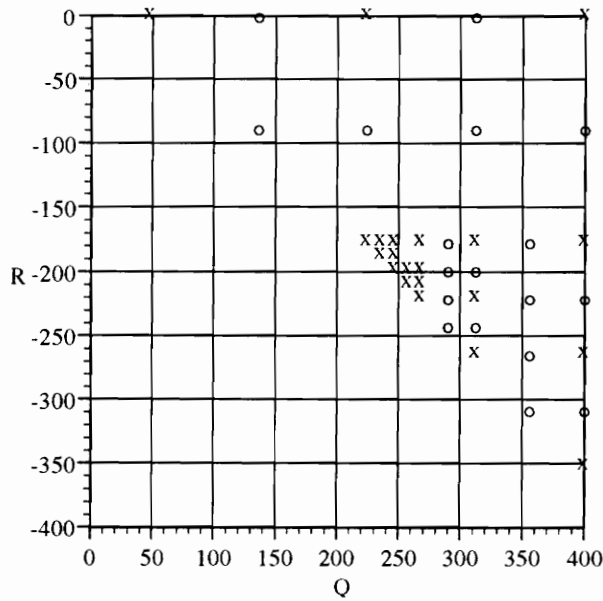
$R = -176$

$TC = \$2.73/\text{day}$.

We have expended 63 computer runs at 21 different points thus far. Figure 4.14a shows a plot of these 21 points in the feasible region. The (Q, R) values found so far compare with the known optimal solution of $(Q^*, R^*) = (194, -130)$ at a total cost of $\$2.58/\text{day}$.



4.14a. Gridpoints where simulation runs are made (X) during the best-first search portion of the search.



4.14b. Gridpoints where simulation runs are made (O) during the first pass of the safety net.

Figure 4.14. The location of simulation runs in Example 2

One pass of the safety-net procedure (our recommended final step of the Starter algorithm), costing an additional 54 runs, yields a new best starting point of:

$$Q = 191$$

$$R = -132$$

$$TC = \$2.72/\text{day}.$$

These additional 18 points are shown in Figure 4.14b.

To be totally cautious and entirely empty the lists \mathcal{E} and \mathcal{D} , the safety-net must be invoked three times more for a total of 113 points (and 339 runs); the same (Q, R) point is obtained as after the first safety net. The 113 total points run under this very cautious Starter strategy is still less than one-third of the 1683 runs at 561 points required if the starter algorithm is abandoned and an exhaustive covering of the region at an inter-gridpoint spacing of $\Delta = 11$ is made.

Example 3: A Multimodal Response Surface

To illustrate that the BFS Starter works even under quite unfavorable conditions, one further example will now be provided. This example shows how the Starter handles a multimodal response surface.

The response employed in this example is based on Crouch et al. (1995) and is a mathematical function with random error added to form the response surface. The particular function and error term used here are

$$y(x_1, x_2) = \begin{cases} 0.5\exp(-2.7r^2)\cos(3\pi r) + 0.5 + \varepsilon & r \leq 0.5 \\ 0.5\exp(-2.7r^2)\cos(\pi r)\cos(4\theta) + 0.5 + \varepsilon & r > 0.5 \end{cases}$$

$$\text{where } \varepsilon \text{ follows } N(0.0, 0.05^2), \quad r = (x_1^2 + x_2^2)^{0.5}, \quad \text{and } \theta = \arctan\left(\frac{x_1}{x_2}\right)$$

A picture of this response over the assumed-to-be user-supplied domain is given in Figure 4.15. Note that this is a very difficult surface to optimize as (1) it has five peaks and one annular depression around the largest peak, and (2) and it has large flat regions (e.g., in the region $x_1 < -0.75$). However, the response surface has relatively low variance. The true optimum of this surface is $x_1 = -0.17$, $x_2 = 0.29$, which yields a maximum of $y = 1.00$.

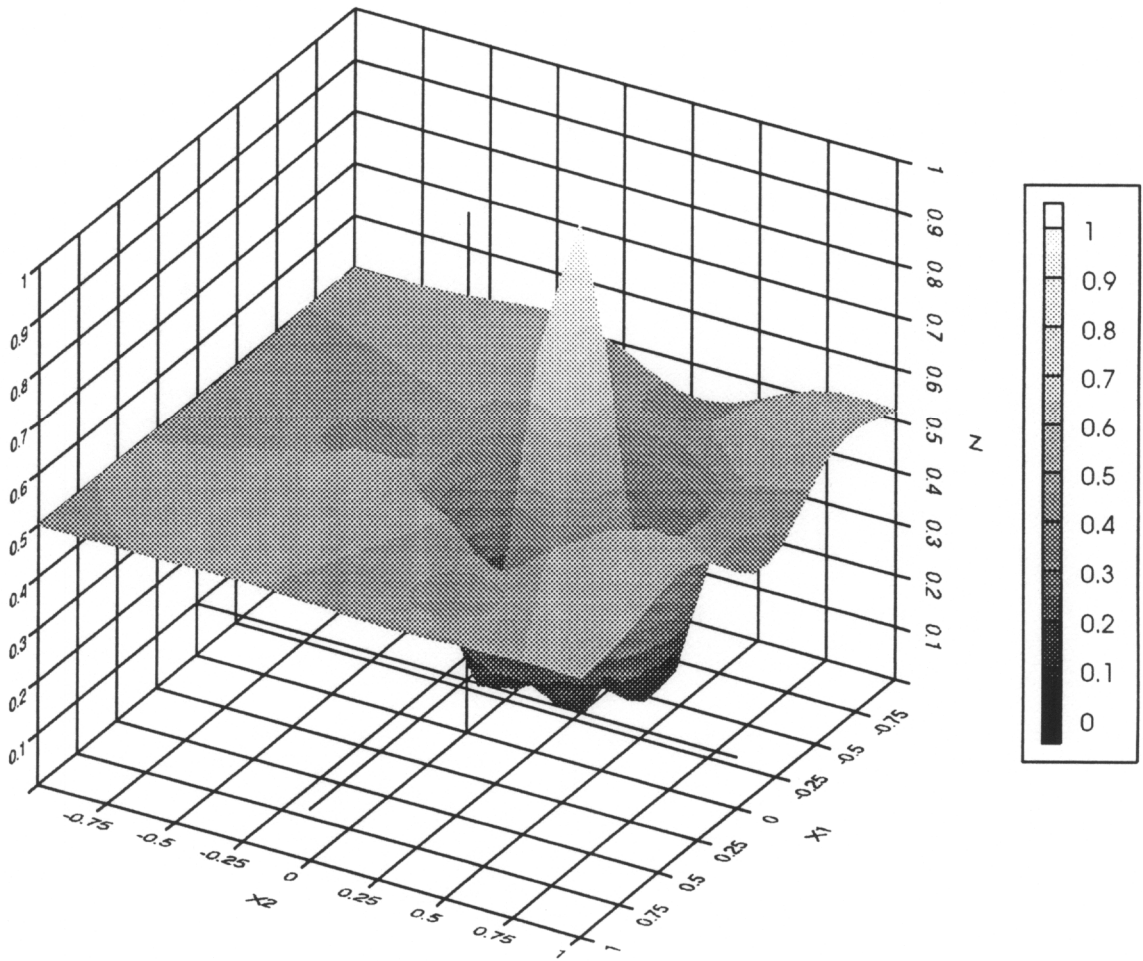


Figure 4.15. The response surface to be optimized in Example 3

The best-first search portion of the Starter yields $x_1 = -0.875$, $x_2 = -0.25$, with $y = 0.5546$. This requires 99 runs at a total of 33 points. One pass of the safety net gives $x_1 = -0.125$, $x_2 = 0.25$, with $y = 0.9378$; this required an additional 87 runs. Observe that this result is very close to the true optimal solution. Clearing \mathcal{E} and \mathcal{D} of all regions requires another 39 runs for a total of 225 runs at 75 points and does not change the solution.

CONCLUSIONS AND FUTURE RESEARCH

This research has defined a Starter for use in those simulation optimization cases where either the starting point or the granularity of the problem are not known in advance. The Starter combines the artificial-intelligence based best-first search with a divide-and-conquer strategy and a safety net.

Three examples have illustrated the Starter procedure. The first example showed that the Starter worked on a “simple” simulation-optimization problem, while the second illustrated the process on a more involved surface with twenty times the variance of the first. The final example represented a very difficult surface to optimize, with multimodal behavior and large flat regions. The Starter worked quite well even without the safety net in all cases, and obtained an optimal solution within 7% after one pass of the safety net in all three cases.

A lower-bound estimate of the number of simulation runs N required by the Starter may be determined based on the original user-specified region and inter-gridpoint minimal spacing. If a best case of the

minimal subdivisions in the divide-and-conquer step occurs, and if no ties occur among the most preferred region at any point in the algorithm and the other regions, then for a square region,

$$N = 12 + 15m$$

$$\text{where } m = \log_2 \left\lceil \frac{\text{range}}{\Delta_u} \right\rceil \text{ and } \lceil \bullet \rceil \text{ is the ceiling function.}$$

For example, if $\Delta_u = 11$ and the user specifies $48 \leq x \leq 400$, then

$$\text{range} = 400 - 48 = 352$$

$$m = \log_2 \left\lceil \frac{352}{11} \right\rceil = \log_2 \lceil 32 \rceil = 5, \text{ and}$$

$$N = 87 \text{ simulation runs.}$$

Conversely, the maximum number of runs required over the specified range to completely cover a square feasible region at a granularity of Δ_u if the Starter approach is ignored is

$$N = 3 \left[\left(\frac{\text{range}}{\Delta_u} \right) + 1 \right]^2$$

In the example above,

$$N = 3 \left[\left(\frac{352}{11} \right) + 1 \right]^2 = 3,267 \text{ simulation runs.}$$

The potential savings using the Starter algorithm is great.

Future work on the Starter involves testing it on more surfaces. The results from these runs can be used to ascertain how aggressiveness may be incorporated dynamically into the Starter process and also to determine which metamodel should be used in estimating the performance of each region, as discussed above.

Chapter Five: Building a Knowledge-Based Simulation Optimization System With Discovery Learning

BACKGROUND

Knowledge-based Simulation Optimization

A simulation model can be thought of as a "black box," with controllable inputs feeding into the box, and the simulation model's responses leaving the box as outputs. The simulation model provides an approximation of how the true system it represents would respond to the given inputs. Each response can be considered to be a function of the inputs with a random error term added.

Figure 5.1 depicts the simulation-model box together with another black box in a feedback loop around it. This second box represents the simulation optimizer. The optimizer takes outputs of the simulation model and uses them to suggest new values for the inputs to the simulation model. The objective of the optimizer is to find inputs that will result in optimal or satisficing responses from the simulation model.

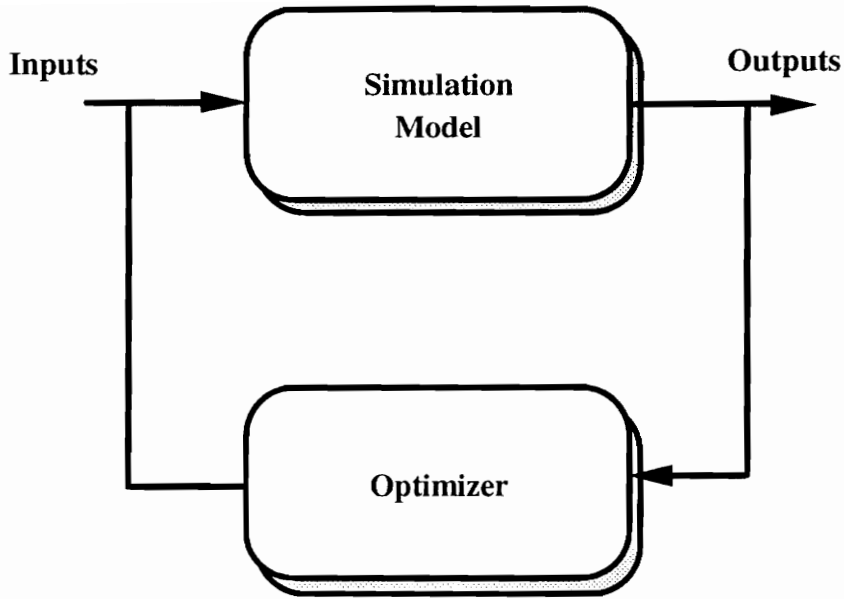


Figure 5.1. The simulation-optimization process

The need for simulation optimization and the costs involved in it have motivated the development of different strategies to search for optimal-response-producing input levels. These strategies range from random and single-factor searches to response surface methodology (RSM) to simulated annealing and genetic algorithms. Meketon (1987) divides simulation optimization strategies into three general categories: nonlinear programming techniques, RSM, and stochastic approximation.

An important decision that must be made in simulation optimization is which search strategy to employ. Some work has been done to aid this decision, although Meketon concludes that "optimization for simulation, to date, remains an art, not a science." He considers the information available (or assumed) about the simulation, and groups optimization methods accordingly to help narrow the choices. Safizadeh (1990) discusses a variety of strategies and their application and concludes that generally RSM approaches are most effective, although some new developments look promising. Smith (1973) performed an empirical study of the effectiveness of several search strategies (random search, single factor search, and four variations of RSM) on a variety of surfaces. He found that the relative effectiveness of each of the

strategies varied depending on the characteristics of the response surface (presence of local optima, random error, number of controllable inputs, etc.).

Surveys of simulation optimization lead to the conclusion that organized guidance is needed to help users choose appropriate search strategies. Safizadeh (1990) explains that: "for successful design and analysis of simulation, one should be well versed in several disciplines." Because of this, users are inhibited from using simulation optimization (and thereby simulation). He concludes that there is, therefore, a need to "develop interactive programs which direct a user to an appropriate optimization technique."

In an earlier paper regarding selection of appropriate optimization technique, Greenwood, Rees, and Crouch (1993) pointed out that there is both art and science in simulation optimization. They further suggested that the art and science should be "separated" in a simulation optimizer, and, in particular, that procedural (e.g., third generation) languages should be used to model the science part, whereas knowledge-based approaches should be used to encapsulate the heuristics that make up the art portion. The particular architecture suggested consists of an inference engine, a knowledge kernel, and processing support modules (see Figure 5.2). The knowledge kernel, in turn, contains three parts: a database to store results, a methodology base to store procedures, and a rule base to store heuristics and to provide control. Note that with this architecture, the fact that optimizer control is resident in the rule base implies that there is no set algorithm for simulation optimization; rather the inference engine (using, for example, backward chaining) can pursue a goal using whatever rules are in the knowledge base. This implies that if the rules are or can be changed, then, in essence, the optimization algorithm itself can change. Exploiting this notion, Greenwood et al. suggested that if results are stored in a database, and if "the algorithm" can be changed by changing rules, then the potential for "doing better" next time, i.e., "learning," exists. This notion of a learner is shown in Figure 5.3. The basic idea is that historical observations are taken from the database in the knowledge kernel of the optimizer, processed by the learner, and then rules are either added, deleted, or changed back in the optimizer rule base. In this manner, not only can heuristics be modified and improved, but so can control of the entire system.

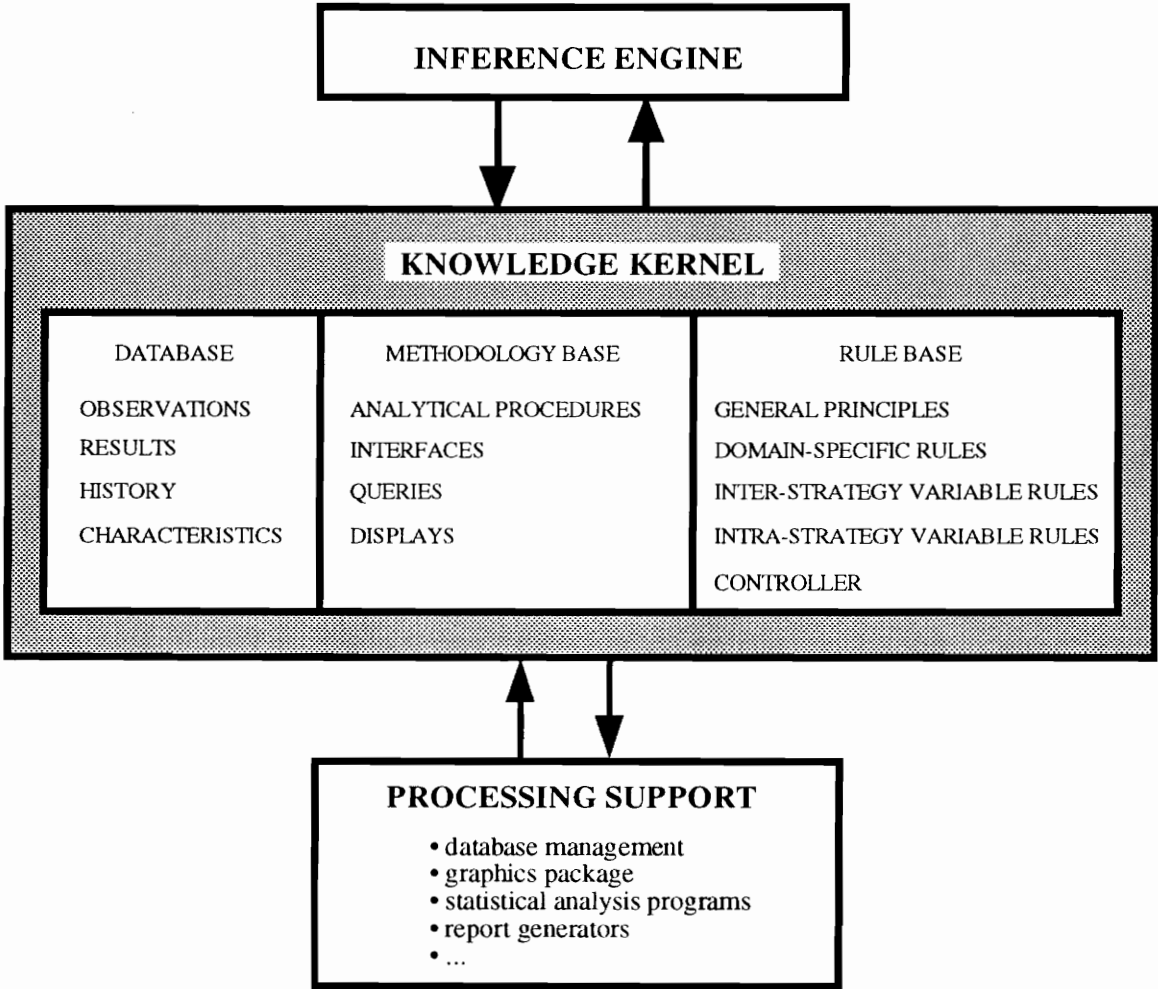


Figure 5.2. Greenwood-Rees-Crouch simulation-optimization architecture

Learning: Definitions, Advantages, and What There is to Learn

Crouch (1992) states that definitions by Simon and Michalski are closest to what she means when she says she will let her optimizer learn. Simon (1983) concludes: "Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same or different tasks drawn from the same population more effectively the next time." Michalski (1986) points out that knowledge acquisition seems

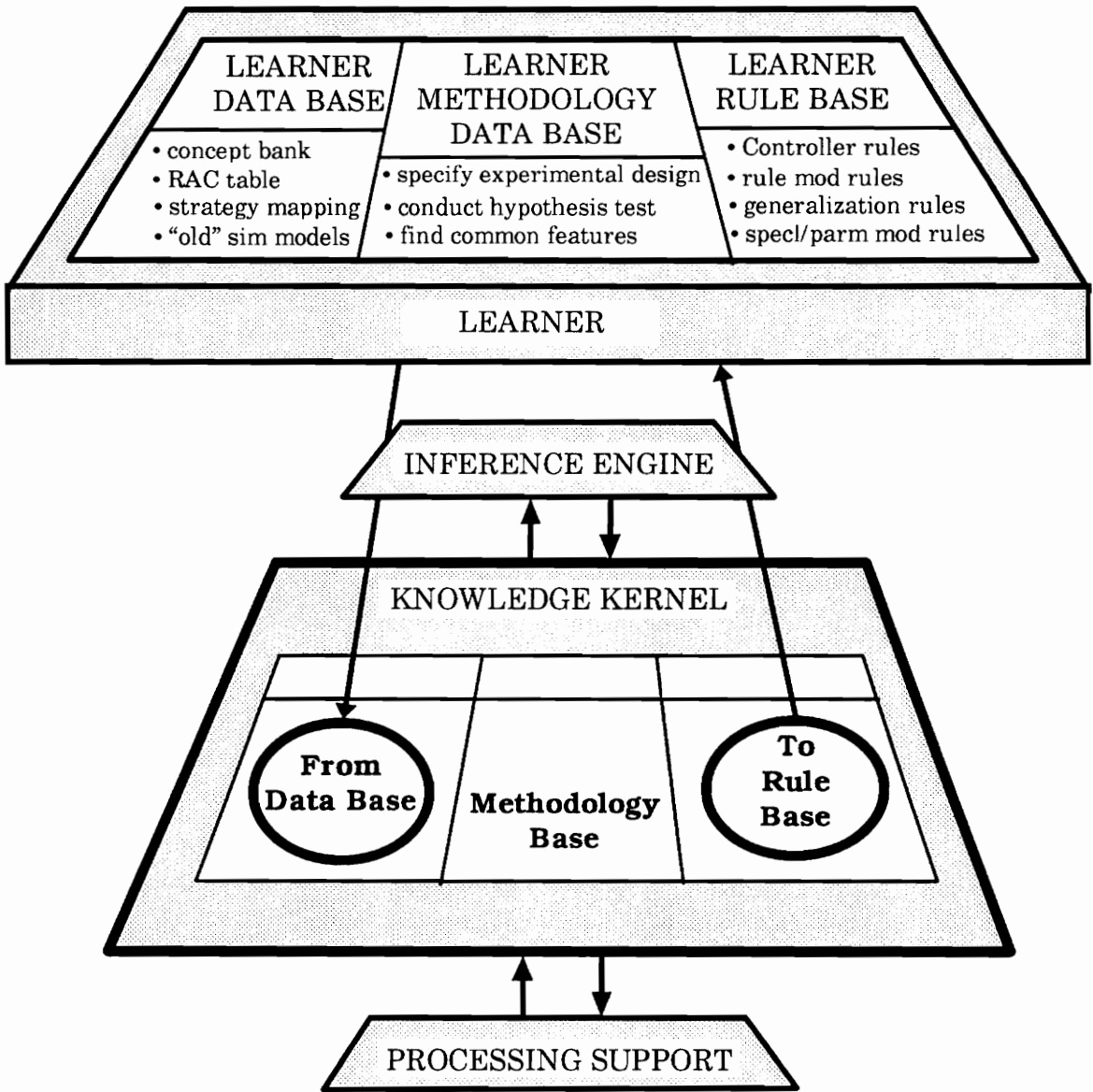


Figure 5.3. Visualization of the Learner and its environs

to be the essence of most learning acts. He adds that in order to acquire knowledge, one has to represent this knowledge in some form. Consequently, he characterizes learning as "constructing or modifying representations of what is being experienced." Thus the optimizer should be able to adapt its performance

so that it improves its optimization on scenarios "close" to what it has already seen. In addition, an optimizer or satisfier with a learning capability should have the capacity to modify or to construct representations of its knowledge, be it knowledge of how to reset certain parameters, knowledge that is domain specific, or knowledge that is more widely applicable as general principles.

Crouch (1992) builds upon a taxonomy developed by Carbonell et al. (1983) to suggest the types of knowledge acquisition a learner should include. The four basic types of learning are (1) rule modification or creation, (2) specialization, (3) parameter modification, and (4) generalization. According to Carbonell, specialization means adding conditions to the "if" part of a rule (the antecedent) so the rule applies to a narrower set of circumstances, and generalization means dropping restrictive conditions in the antecedent to make the rule apply in a wider variety of contexts. By parameter modification is meant the changing of a numerical value in a rule; for example, the antecedent "IF number of runs > 12" could be changed to "IF number of runs > 10." Rule modification results in changing the consequent of a rule. For instance, a current rule may conclude that RSM is the preferred search strategy ("...THEN strategy = RSM"); however, learning may suggest that simulated annealing is preferred. Thus, the modified rule would have the consequent "THEN strategy = simulated annealing."

In this research, we will limit ourselves to the four types of learning just elaborated, noting that additional types of learning can be added to the Learner later if desired as plug-in modules.

What it is that can be learned in a simulation optimization system with these four types of learning has been pointed out in Crouch (1992). In order to understand these ideas, however, it is first necessary to present a quick overview of CGR's (1995) "Classifier KBSOS." CGR called their system a "Classifier KBSOS" because its simulation output surfaces are *classified* according to the search strategy most likely to render success.

In the Classifier KBSOS, input sufficient to define the problem is obtained from the user in the User module (see Figure 5.4). This input is then fed to the Classifier module, where three steps occur. First, the "shotgun" suggests an initial set of simulation runs to be made at various input combinations across the surface. The results from these computer runs are then input to the "synthesizer," which attempts to develop a fitted or synthesized surface through those points. (A neural network can be and was successfully used for this by Crouch et al. The reason for this synthesis is that it hopefully will save computer runs by characterizing the synthesized or estimated surface rather than depending entirely on actual runs.) Then the synthesized surface is analyzed by several procedural programs and heuristics in the "characterize" module in order to classify or characterize the response surface. The idea of classifying a surface is based on a study reported by Smith (1973) in *Operations Research* in 1973, which found that optimal search technique varies by type of surface. Crouch et al. used the same explanatory variables Smith used in his study to classify their surfaces with the Classifier KBSOS.

Once a surface has been classified, rules in the KBSOS knowledge kernel invoke the Strategy Selector. This module is a collection of rules that choose a search strategy (e.g., RSM, random search) depending on the surface characteristics identified by the Classifier. Note that as the whole classify-and-select-strategy process is iterative, additional search may result in reclassification of the surface and hence specification of a different strategy as the optimization proceeds. After a search strategy has been chosen, the Strategy Detailer (another set of rules) is fired, and implementation particulars are set whereby the Search may be conducted.

As Crouch points out, it should be clearly stated what is not meant when one suggests that a KBSOS will learn. The learner is not expected automatically to derive or infer a never-before-seen search technique whenever a previously unanalyzed surface is encountered. Rather, the learner is expected to perform such tasks as to modify parameters in the shotgun, to suggest that a new antecedent be included in a set of rules in the Strategy Selector, or to respecify the number of runs to be made at the center point of a given search

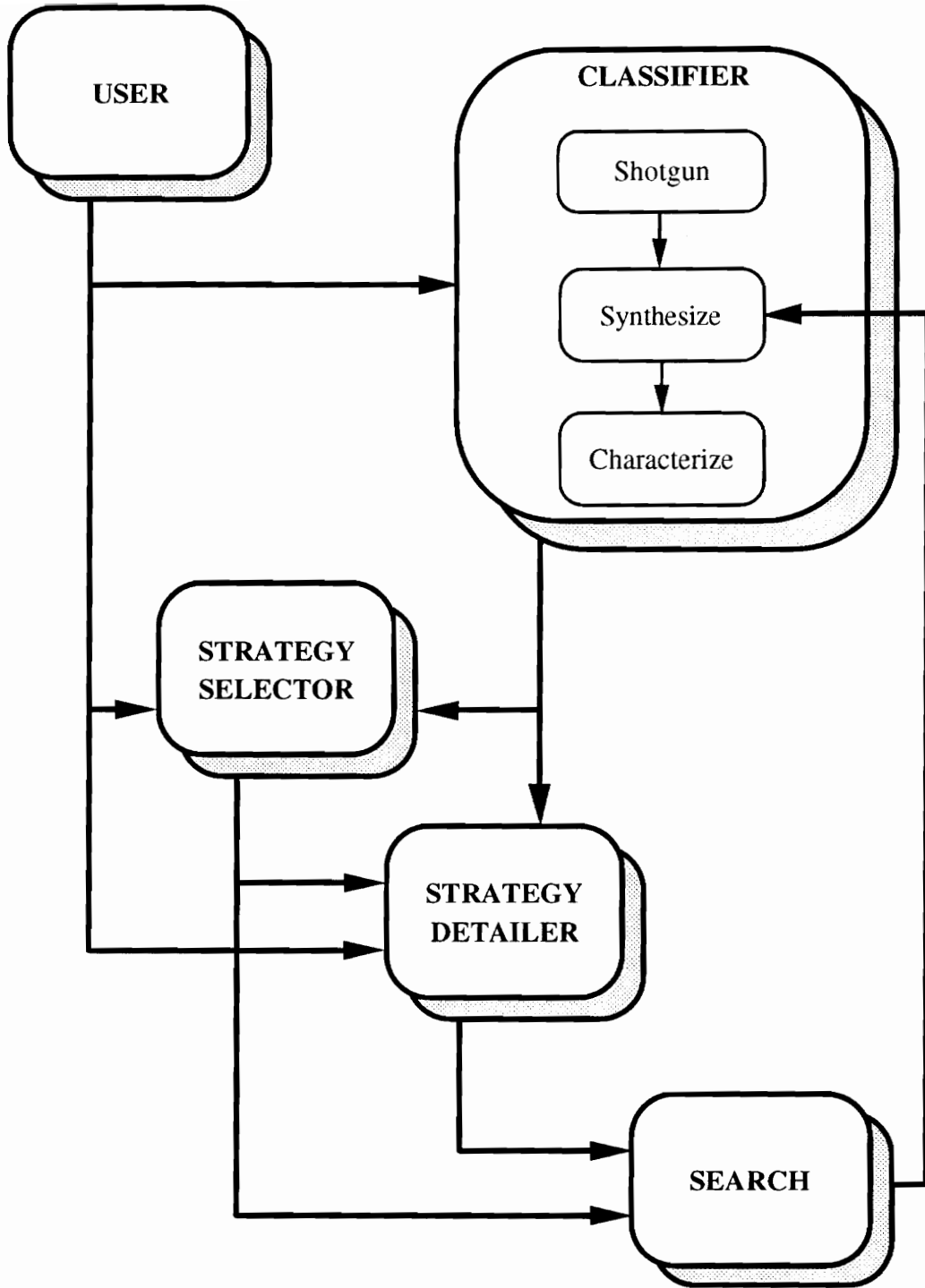


Figure 5.4. An overview of the Classifier KBSOS

being implemented. Learning is to be incremental as opposed to far reaching, and it will only be successful as its databases of surfaces and experiments grow large.

In order to indicate how learning will take place in a KBSOS, Crouch (1992) lists some examples of each of the four kinds of learning; see that reference and Crouch, Greenwood, and Rees (1995) for further details:

parameter modification: - in the Classifier: re-specifying the number of runs to be made randomly and at regular grid points in the shotgun module; re-setting a variance threshold, above which additional replications of data points used to fit the synthesized surface will be collected; re-stipulating the vertical distance *delta* from the true optimum, within which non-adjacent portions of the response surface indicate multiple, optimal solutions. And in the Strategy Detailer, re-adjusting the step size for a given search technique.

specialization: - adding new concepts as antecedents to the rules in the Strategy Selector (e.g., adding "IF variance is not high" to a current rule specifying RSM as the search procedure); adding a similar clause again to the IF part of an existing rule in the Strategy Detailer (e.g., adding "IF lack of fit is significant" to a rule specifying a shift from a first- to a second-order RSM design).

rule modification: - in the Strategy Selector, if some cases concluding in "THEN Strategy = S_1 " achieve different levels of success than others, then separate these cases and respecify "THEN Strategy = S_2 ," a new strategy whereby there is some evidence that S_2 will work better on the poorer cases than S_1 did.

generalization: - deleting existing concepts from the antecedents of rules when there is evidence that such concepts are irrelevant to the Strategy Selection being made (e.g., removing "IF distance to optimum = far" from a rule concluding in "THEN Search = random search.") Generalization is also helpful in a housecleaning sense in that rules can at times be combined, thereby reducing the number of rules in the rule base.

It is easily noted from the above lists that there are a plethora of details to be learned; this is because, fundamentally, so much of simulation optimization is heuristic, or "art." The approach taken in Crouch (1992) and that we have taken here is to prioritize what we want to learn with our KBSOS. We have placed the Strategy Selector as our top learning objective, with its specialization, rule modification, and generalization. At second priority is the Classifier, which calls primarily for parameter modification learning.

Having examined the Classifier KBSOS, definitions of learning, and what it is that may be learned in a knowledge-based simulation optimization system, we now direct our attention to the Crouch (1992) Learner. This will provide the final building block needed to explain the Learner we have actually constructed ourselves.

The Crouch Learner

Overview

Each of the four learning types to be included in Crouch's learner requires both procedural and heuristic computation. That is, each learning type consists of both procedural decisions such as hypothesis testing that can best be performed by algorithmic means, as well as heuristic processing best done in, for example, knowledge-based systems. A major design decision made by Crouch was to separate the "art" and "science" in the learner, just as Crouch, Greenwood, and Rees (1995) did in the KBSOS.

Figure 5.3 shows Crouch's learner sitting above the KBSOS and deriving input from the KBSOS database; changes are passed back to the KBSOS rule base. Figure 5.3 explicitly illustrates the implementation of the separation of art and science in the learner in terms of its three modules, the Learner Data Base, the Learner Methodology Base, and the Learner Rule Base. In addition, Figure 5.3 shows some of the functions to be carried out by each of the three modules.

According to Crouch (1992), a knowledge-based simulation optimization system contains many concepts that may be stored in a variety of representation formats, including tables, rules, and neural networks. In order to be able to manipulate this information in a learner, the Learner Data Base must keep a registry of concepts and their interrelationships. Crouch's mechanism for doing this is a concept bank and a Relationships Among Concepts (RAC) table. The RAC table stores which concepts are used in which

rules. As indicated in Figure 5.3, both the concept bank and RAC table are (important) components of the Learner Data Base, as is the strategy mapping, which will be described later in this chapter. An additional item included in Crouch's Learner Data Base is a collection of "old" simulation programs. That is, she suggested that whenever a simulation program was run and its results were stored in the database, it would be advantageous if the program (i.e., the code) itself were left in a library in the Learner Data Base, in case the Learner decided later to do further exploration with the program. Obviously, this is not necessarily practical in all cases. But the more the Learner has access to in the way of history, the more likely it is to be successful. Finally note that Crouch's Learner Data Base may share or coincide or differ from the knowledge kernel data base.

The Learner Methodology Base consists of whatever procedural aspects are necessary to implement the four types of learning. For example, if the Learner were investigating the advantages of changing a troublesome parameter, it might decide to conduct an experiment to test the proposed change. In such a case, the Learner would call the experimental design submodule, which would specify where computer runs should be made to carry out (say) a fractional factorial design. Then a second submodule in the methodology base, a hypotheses testing procedure, would evaluate the results of these experiments to determine statistically the worth of the change. Crouch admits that these submodules may be complex, but add that they can be implemented using ideas well-established in the literature. A third submodule in the learner methodology base deals with searching for common features or concepts for a given set of rules.

Crouch's Learner Rule Base contains all the rules or heuristics needed to do specialization, rule modification, parameter modification, and generalization. Moreover, it also possesses a set of controller rules, which decide when to invoke each of the four learning types. All of these rules, under the direction of an inference engine, drive the Learner in its search for an improved simulation optimization process, and call the Learner Data Base and Methodology Base when needed.

Crouch Process Flow

A brief overview is now given of the Crouch learning process flow; details may be found in Crouch (1992). This process is based on Slade's work on case-based reasoning (1991). Slade never examined the simulation optimization context; rather Crouch adapted some of the basic concepts in case-based reasoning and learning and modified them for this application.

Figure 5.5 indicates the flow of Crouch's learning process. The shaded boxes indicate the major operations in the process needed for all four learning types. (The only exception is that Repair is not needed in Generalization learning.) The learning process for any of the types begins with Retrieve, where learner rules are used to extract relevant data from either the learner or knowledge kernel databases. Upon retrieval, learner modification rules are invoked to suggest changes in some aspect of knowledge kernel rules. This occurs in the Modify block. For example, in parameter-modification learning, a particular parameter is suggested for change; whereas in specialization learning, retrieved data cases are first segmented by performance, and concepts in the antecedents are then sought that can explain the performance differences. Once a modification is proposed that hopefully improves KBSOS performance, the Test block is called. Basically, the Test block determines whether the proposed modification results in an improved solution (i.e., a new set of rules), or rather in no improvement or possibly failure. In the first case, control passes to the Assign and Store blocks, where the proposed modifications are actually made and put back in the KBSOS rule base. In the case of failure or no improvement, the Explain and Repair blocks are called, where either abandonment of learning for this case occurs due to unsuccessful explanation and repair, or further modification leads to a successful solution. This latter case leads back to assignment and storage, as Figure 5.5 indicates.

Although Crouch's research has suggested an architecture and a learner flow, details were not specified as to how all modules would work for the four types of learning. Moreover, since a Learner has never

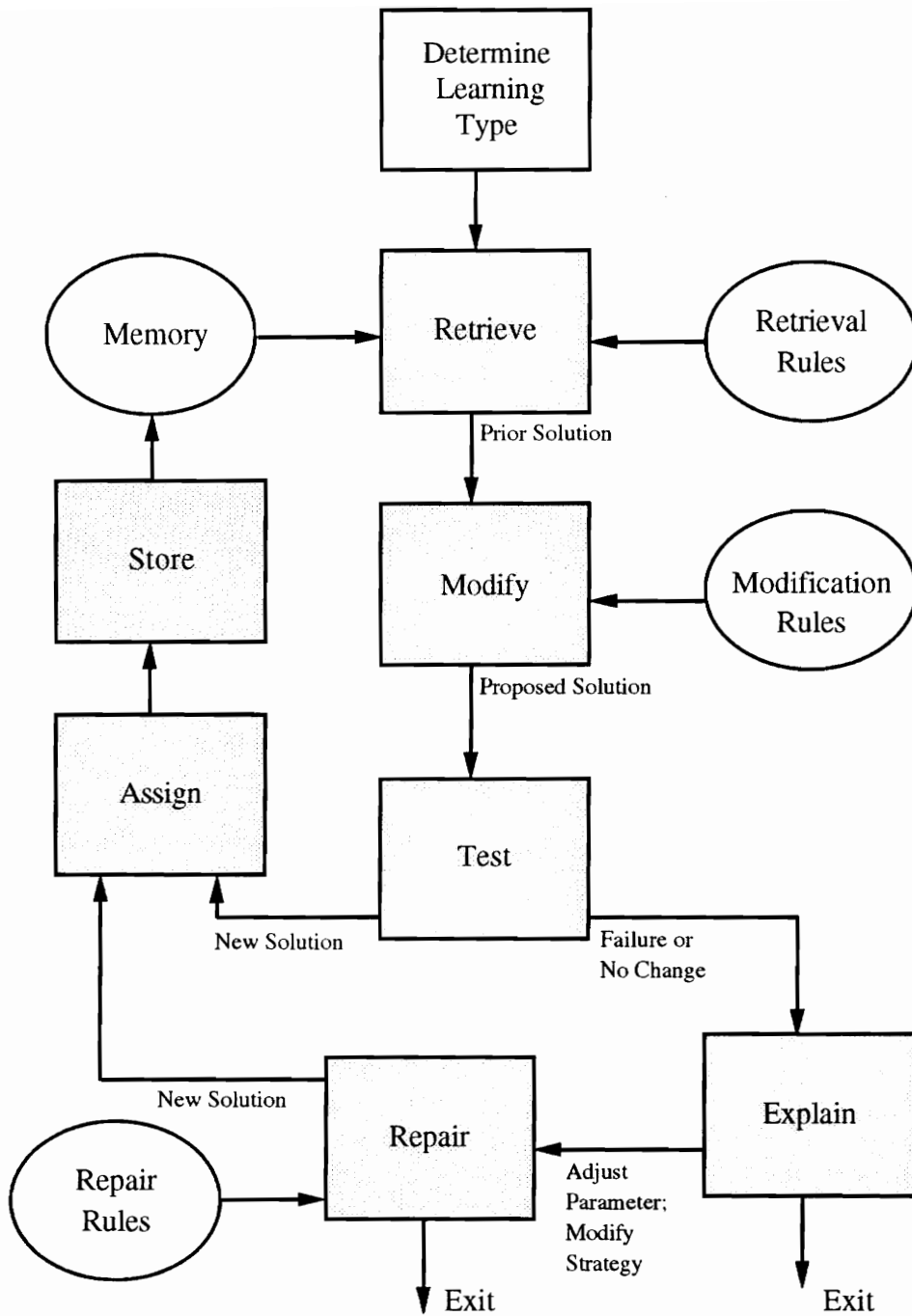


Figure 5.5. Crouch's learning process (Crouch, 1992)

been built, it is not known whether such a Learner is truly practical. The research described in this chapter specifically addresses these issues, making three contributions. First of all, we build a Learner and test it on a simulation example. Second, having successfully constructed a Learner, we are able to specify an architecture and process flow; in particular, it will be seen that a clear explanation of how discovery takes place was not provided in the Crouch paper. And finally, an analysis of what must be done next to extend the Learner to larger-scale, more complex scenarios is described.

The remainder of this chapter is organized as follows. The next section describes a general model of "discovery," and the following segment details the modified general learning flow of our discovery learning system. It will be found that the Crouch (1992) architecture of Figure 5.3 contains most of the components necessary in a Learner, but is lacking in clear explanation of how discovery will take place -- in particular how domain knowledge and search will be used in this process. This discussion is followed in turn by a detailed inventory simulation example invoking the Learner illustrating parameter modification. The chapter concludes with a summary and a description of future steps.

KNOWLEDGE DISCOVERY

The use of knowledge discovery concepts in a knowledge-base simulation optimization system (KBSOS) is new. The previous work in KBSOS did outline and define four kinds of learning which required storing some items in the form of a database. It is a simple extension then to use knowledge extraction techniques for databases in an attempt to learn something from the data being stored.

The essence of learning as we use it here is knowledge discovery. Frawley, Piatetsky-Shapiro, and Matheus (1992) present a prototypical framework for knowledge discovery under a different setting than simulation optimization, namely databases. This framework is redrawn in Figure 5.6; it contains five components (besides the discovered knowledge itself). Since our research builds a Learner based upon both the Frawley et al. paradigm and the Crouch (1992) architecture and flow, we now discuss the former in some detail.

The Frawley discovery system has as its core the discovery method, which computes and evaluates patterns on their way to becoming knowledge. Note in Figure 5.6 that the discovery method has two principle components: search and evaluation. Inputs to the discovery method include the database itself, its data dictionary (which defines field names, the allowable data types for field values, various constraints on field values, etc.), additional domain or background knowledge, and a set of user-defined biases that provide high-level focus. The output of the discovery method, of course, is discovered knowledge that can be directed to the user and/or fed back into the system as new domain knowledge. Frawley et al. note that both the user bias and the domain knowledge assist discovery by focusing search; i.e., these sources guide and constrain search by, for example, telling a system what to look for and where to look for it. These constraining influences are both desirable and undesirable: the former in that discovery is made easier, and the latter in that valuable discovery may be ruled out by the constraints.

Frawley et al. (1992) point out that discovery algorithms inherently contain two processes: identifying interesting patterns and then describing them in a concise and meaningful manner. They note that the identification problem is essentially a problem of pattern identification or clustering, which in essence is the problem of finding classes such that the similarity within classes is maximized while the similarity among classes is minimized. For example, it might be important for a firm to discover that the major purchasers of its product is a particular set of individuals, whereas other individuals tend to have very little interest. Concept description involves the summarization of relevant qualities of the pattern classes

rather than just enumerating them. For example, it would help the firm described above to know that the particular set of individuals is the class of white males between the ages of 15 and 20. According to Frawley, well-known approaches to concept description include decision-tree inducers (Quinlan, (1986)), neural networks (Rumelhart and McClelland, (1986)), and genetic algorithms (Holland et al., (1986)).

KNOWLEDGE DISCOVERY IN THE SIMULATION OPTIMIZATION DOMAIN

Figure 5.7 illustrates the architecture of our Discovery Learner for simulation optimization and its interaction with the Classifier knowledge-based simulation optimization system. The Classifier KBSOS, shown at the right in that figure, contains three principle modules: an inference engine; a knowledge kernel, which contains the rules and algorithms necessary for simulation optimization, as well as a record (a database) of the optimization session; and processing support, including interfaces to users, the simulation program, etc. Crouch, Greenwood, and Rees (1995) may be seen for further details on the Classifier KBSOS.

The Learner, shown as an "L"-shape at the left of Figure 5.7, contains the same modules as the Frawley et al. paradigm, but is adapted to fit the purposes of the simulation-optimization environment. These modules are the sessions history database, the data dictionary, a domain-knowledge module, and (at its heart), the discovery-methods module. As in Frawley, bias is provided to the Learner from a user/developer.

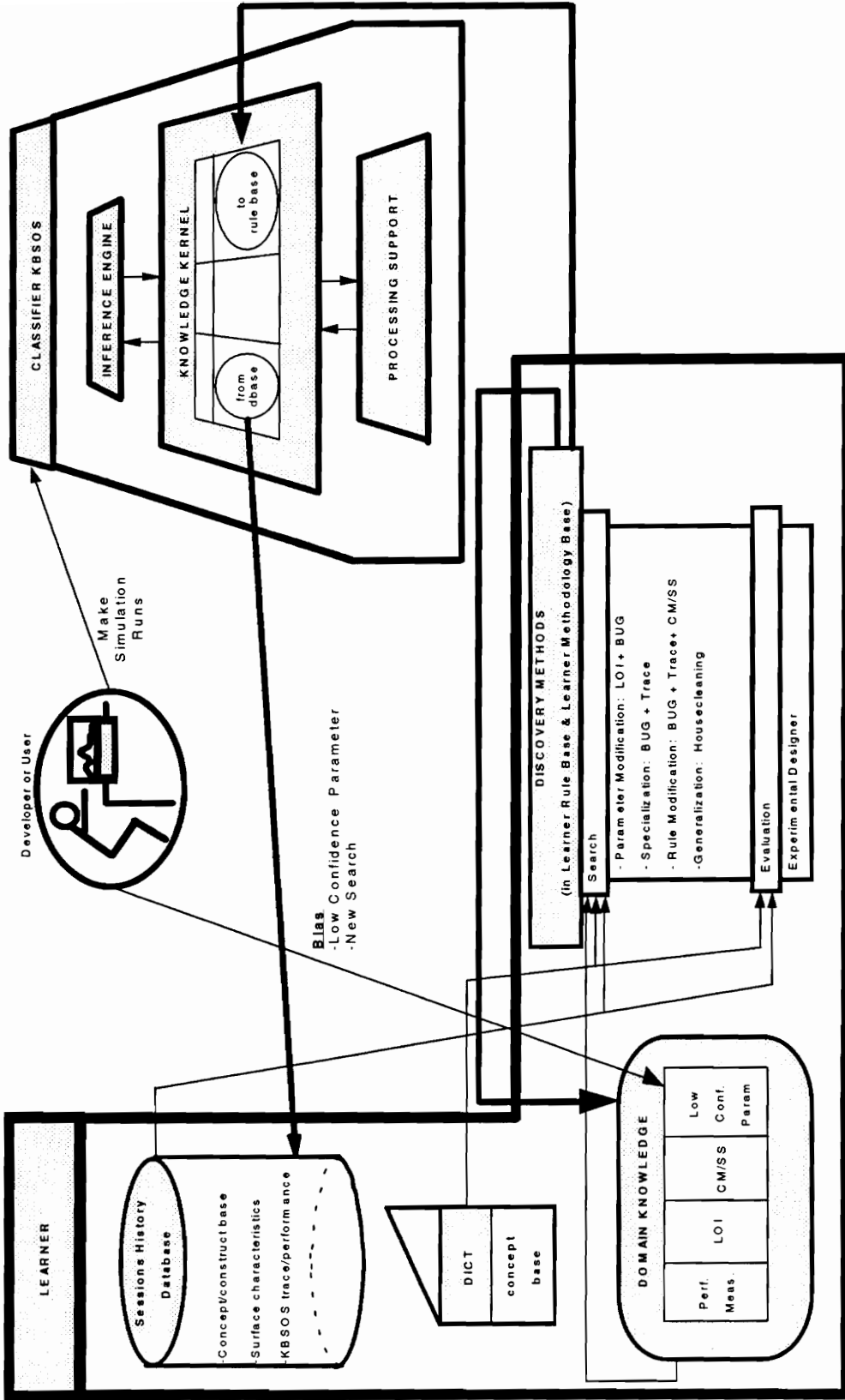


Figure 5.7. The Discovery Learner & its interaction with the Classifier KBSOS

Note that the key information/knowledge flows between the Classifier KBSOS and the Learner consist of one primary flow from the KBSOS to the Learner, and two flows from the discovery-methods module: one back to the KBSOS, and another internal to the Learner, back to the domain-knowledge module. These three flows are emphasized in Figure 5.7 by the heavier lines and arrows. The key notion is that information from optimization sessions (stored in the database of the knowledge kernel) flows to the Learner as input where it is recorded in the Sessions History Database. Similarly, what is learned by the Learner flows back as output to the rule base of the KBSOS, so that rules are modified; consequently, simulations conducted in the future by the KBSOS will (hopefully) be improved. What is learned by the Learner also flows back to the domain-knowledge module in the Learner, as a means of keeping the Learner up-to-date. These flows constitute the primary activity of the Learner, with all other activities conducted in support of that activity. We now detail this support, proceeding module-by-module through the Learner.

Data Dictionary

The data dictionary maintains the concept bank, namely a list of concepts or constructs utilized in the sessions history database. For example, some of the concepts in our sessions base are number of controllable factors, distance from the optimum, level of factor activity, and presence of local optima. The concept bank also contains, as mentioned, allowable data types for field values as well as any constraints on field values. The data dictionary employed in our Learner is not significantly different from data dictionaries employed in other applications.

Sessions History Database

The Learner database is called a Sessions History database because it records the history of sessions carried out by the KBSOS. There are three kinds of information regarding any session maintained in the database, each carried to meet a different need for the Learner. The first is the concepts and the values that each can take. The second is a description of session characteristics which includes a session trace, the search method and results, surface characteristics, and the activating rule, among others. The third kind is a detailing of the rules including parameters and associated levels. Figure 5.8 is a lattice that shows some of the relationships between the three kinds of information.

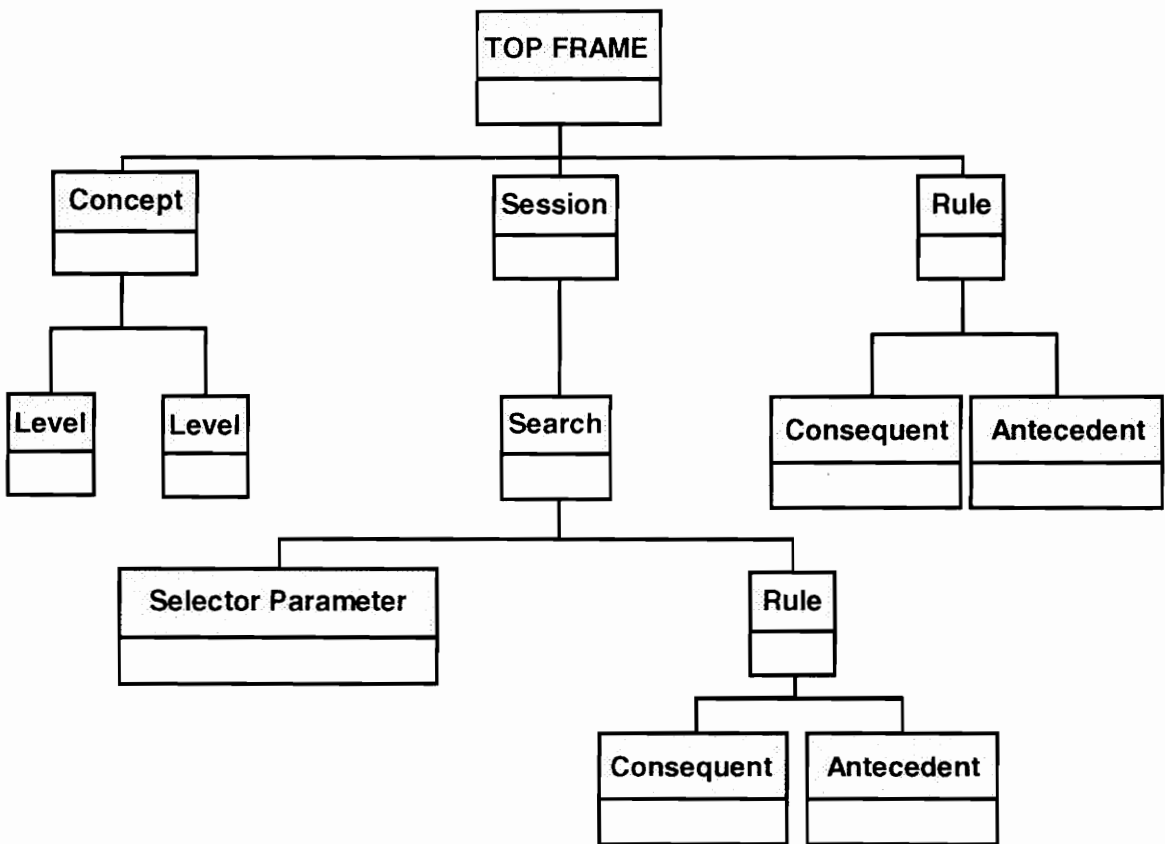
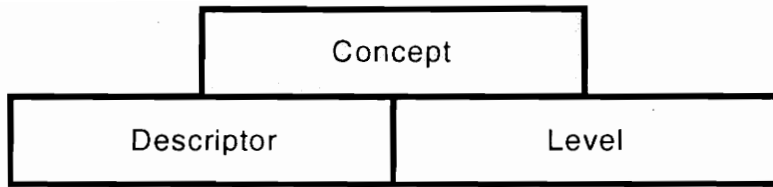


Figure 5.8. A lattice showing the interconnections of the Sessions History Database frames

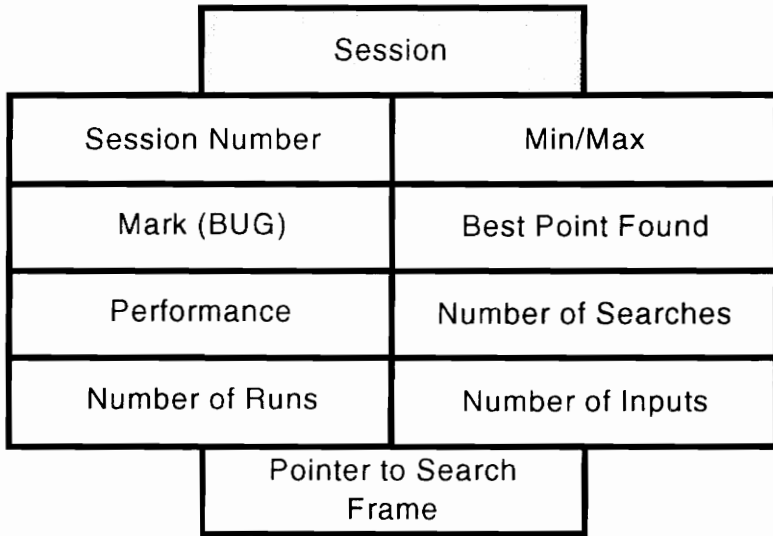
All three kinds of information are represented as frames. The concept frame (see Figure 5.9a) contains the name of the concept and the possible values that the concept can take. The concept frame can be a child frame of an antecedent or consequent frame. The session frame (Figure 5.9b) contains session specific information such as the session number, the goal (min/max), performance, number of searches performed, a rating of effectiveness, total number of runs, number of inputs, and the best solution found. It has one child frame called the search frame. The search frame contains search specific information such as number of runs used, search method, best point found and the surface characteristics as estimated at that point (the selector parameter in Figure 5.8). The search frame has three child frames, (1) the activating rule frame, (2) the trace frame, and (3) another search frame if an additional search had been performed (the value is null if no additional searches were performed). The rule frame (Figure 5.9c) contains the rule name, the rule base it belongs to and two child frames; antecedent and consequent. The antecedent and consequent frames have pointers to concept frames and logical operator slot. A trace frame contains the points visited.

Domain-Knowledge Module

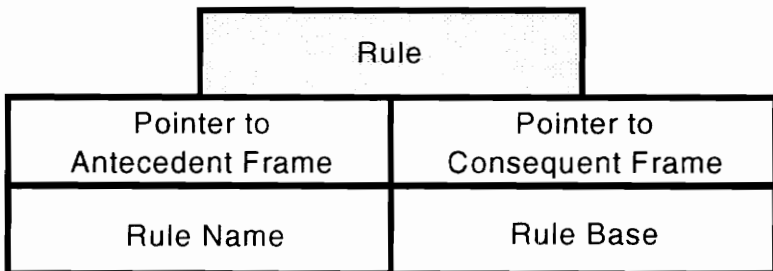
The third component of our Discovery Learner is the Domain-Knowledge module. As mentioned, discovery must often be focused if the knowledge discovered is to be useful, and sometimes it must be so if there is to be any discovery at all. The general purpose of the Domain-Knowledge module is to enable the discovery that occurs in our Learner to be relevant and useful to the Classifier KBSOS. In particular, the function of the domain-knowledge component is to provide guidance to the search portion of the Discovery Methods module in four particular ways, one for each type of learning: (1) what parameters can/should be considered for modification (this is for parameter-modification learning), (2) which rules are candidates for specialization, (3) which rules should be modified in their conclusions (e.g., recommending different search strategies for rule-modification learning), and (4) when to attempt generalization.



5.9a. Concepts Frame



5.9b. Session Frame



5.9c. Rule Frame

Figure 5.9. Examples of frames

Of course, there is a danger in providing domain knowledge to our system in that specifying such knowledge can rule out potentially valuable discovery. Frawley et al. (1992) point out the case in logistics planning where the search space is so large that it is impossible to find solutions without using constraints such as "trucks don't drive on top of water (without bridges)." But adding this constraint eliminates potentially interesting solutions such as those in which trucks drive over frozen lakes in winter. So the key, they say, is to provide as general as possible constraints, while still maintaining enough specificity to provide useful discoveries. We have tried to walk this "fine line" in our Domain-Knowledge module.

There are four primary components in the Domain-Knowledge module; these may be modified or enhanced in the future. They are

- the performance measures component
- the low-confidence parameter list
- the link-of-influence submodule, and
- the classifier-methodology-to-search-space (CM/SS) component.

We now describe each of these components.

The performance-measures component contains the currently recommended measures for evaluating success in the KBSOS. At this point, we are utilizing the same performance measures as Crouch (1992), not because we have studied them and found them acceptable, but rather because we have focused our efforts elsewhere and have assumed them by default. (We believe this whole area to be a topic worthy of further study.) There are two Crouch performance levels, weak and strong, and both are defined in terms of what Crouch called "interesting" optimization sessions or cases. Two of Crouch's three "interesting" cases are oriented toward the efficiency of the optimization, which Crouch measured according to the total number of runs used to find the optimal response. Those optimization sessions requiring relatively many

runs are marked "Bad" or "B," whereas those requiring relatively few runs are marked "Good" or "G." The other Crouch "interesting" case is based upon effectiveness, which she measured by observing the variance of the surface and whether multiple optima exist. If there is high variance or if multiple optima exist, Crouch labels the case "Ugly" or "U." We refer to Crouch's three interesting cases as "BUG."

As mentioned, Crouch then defined performance in terms of the BUG cases. Performance is judged as "strong" or "weak" according to the following two (Crouch) rules:

IF marked = G AND

marked < > U

THEN performance = strong;

IF marked = B

THEN performance = poor.

The performance measures "strong" and "weak" are used in the Discovery Methods module as will be explained shortly. With the modular structure of the domain-knowledge module, it is relatively easy to modify performance measures as desired.

Again, it is the purpose of the first of the four Domain-Knowledge module components, namely the performance-measures component, to provide the criteria whereby the success and failure of the KBSOS may be judged.

The second component in the Domain-Knowledge module is the Low-Confidence Parameter List. This list is simply a developer-supplied tabulation of the "important" parameters utilized in the rules. They are ranked according to the lack of confidence the developer has in their values, with least-confidence parameters at the top of the stack. When the Learner decides to attempt parameter modification, it will do so by popping the low-confidence-parameter-list stack, and considering the modification of the parameter

at the top of that list using the parameter modification process flow outlined in Crouch (1992). Figure 5.10 shows where the KBSOS parameters that can be modified are located within the Classifier KBSOS.

The third aspect of the Domain-Knowledge module is the link-of-influence (LOI) submodule. The basic purpose of this component is to establish the link between any parameters to be modified and the effect on rules “downstream” in the knowledge base. For example, assume a given parameter in the “characterize” component in the classifier module in the KBSOS is presently set to a value of 0.5. If a change to 0.7 for this parameter is under consideration, then those cases (i.e., sessions) for which the parameter took on values between 0.5 and 0.7 must be re-examined. Now if the parameter being set at 0.7 in the “characterizer” caused a particular rule in the Strategy Selector to be fired and another rule in the Detailer subsequently to be fired, then the effect of the change to 0.7 must be considered to the extent that the downstream rules in the Selector and Detailer that would be fired instead of the initial set must be examined. For instance, the change from 0.5 to 0.7 might result in a whole new search strategy being chosen in the Selector.

The determination of the downstream rules affected by a parameter shift is not difficult conceptually, as one merely needs to forward chain through the rules. Figure 5.11a shows how this works with a few rules and five sessions. The parameter α affects the parameter β which in turn affects the parameters γ . The parameter γ affects the number of replications but only for one search method. By forward chaining through the rules the parameters that are affected can be found. We have written such a domain-specific forward chainer and placed it in what we call the “link-of-influence” submodule, since the chaining establishes the influential links in the connection between any parameter and the rules impacted. The threshold for parameter α is set at 0.5 and the rules that are affected by α are shown in Figure 5.11a. The effect of changing the threshold from 0.5 to 0.7 is shown in Figure 5.11b. Note that only two sessions (3 and 4) are impacted by the change. The particular modules in the Classifier KBSOS affected by the LOI submodule are also shown in Figure 5.10 by the dashed lines leading from that submodule.

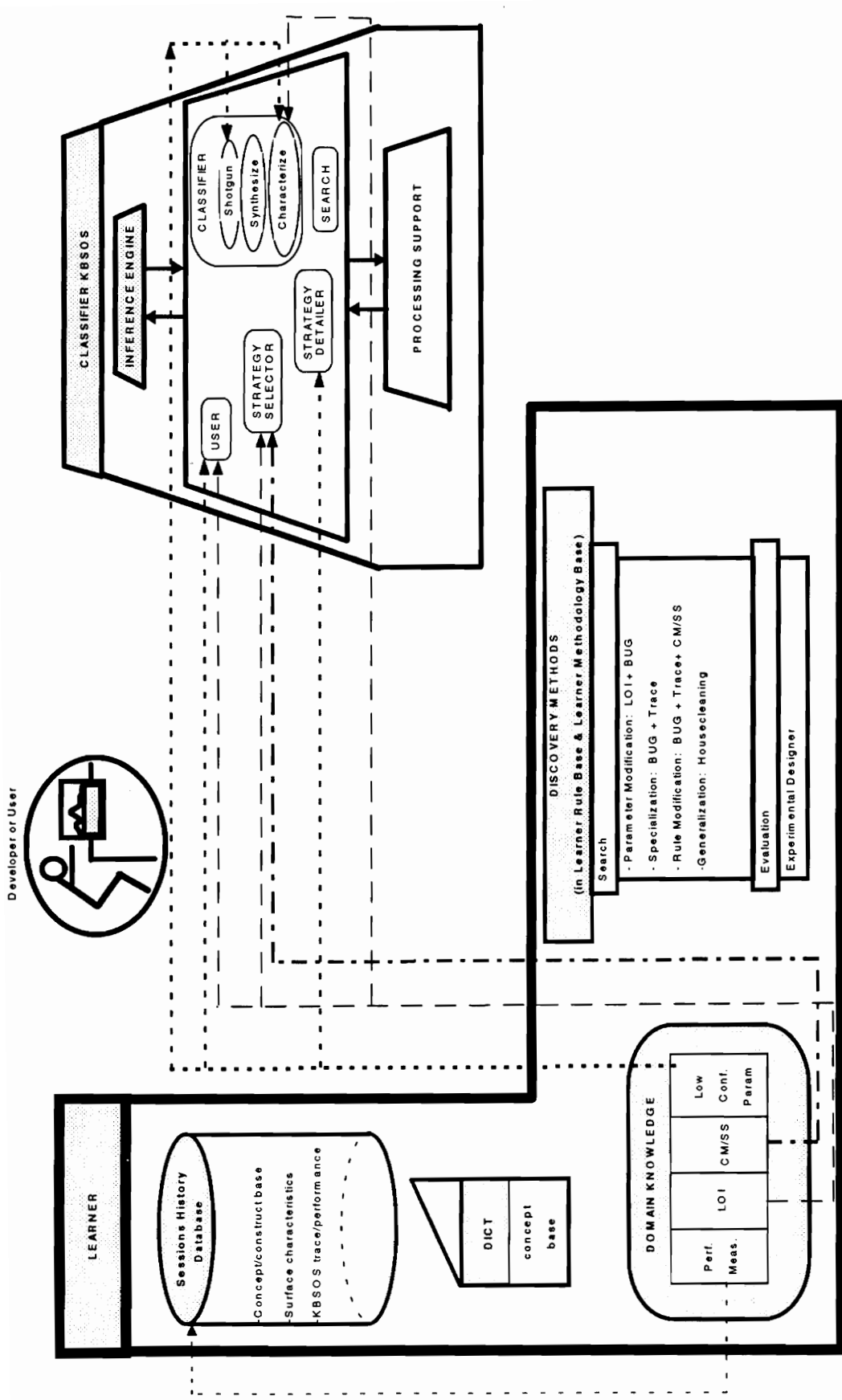
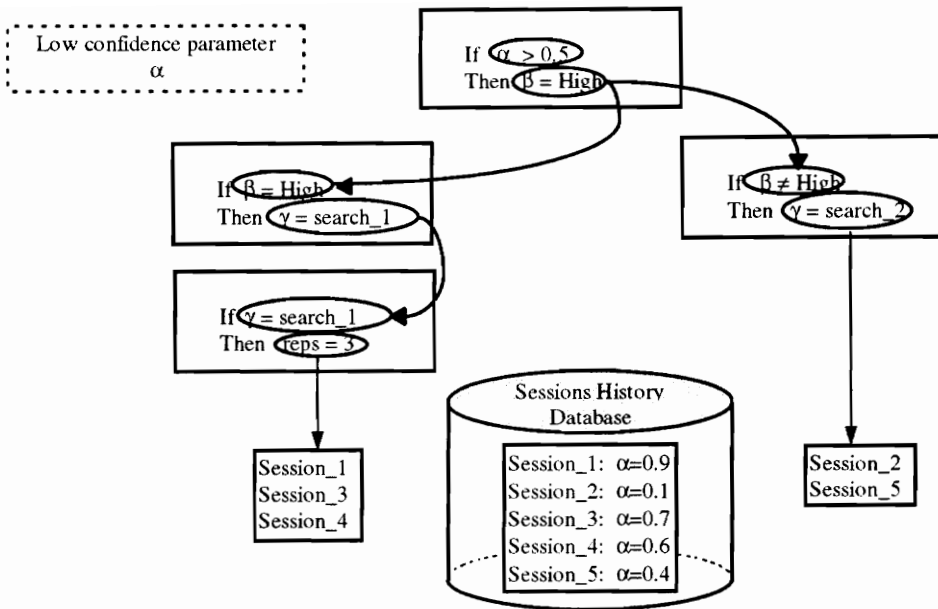
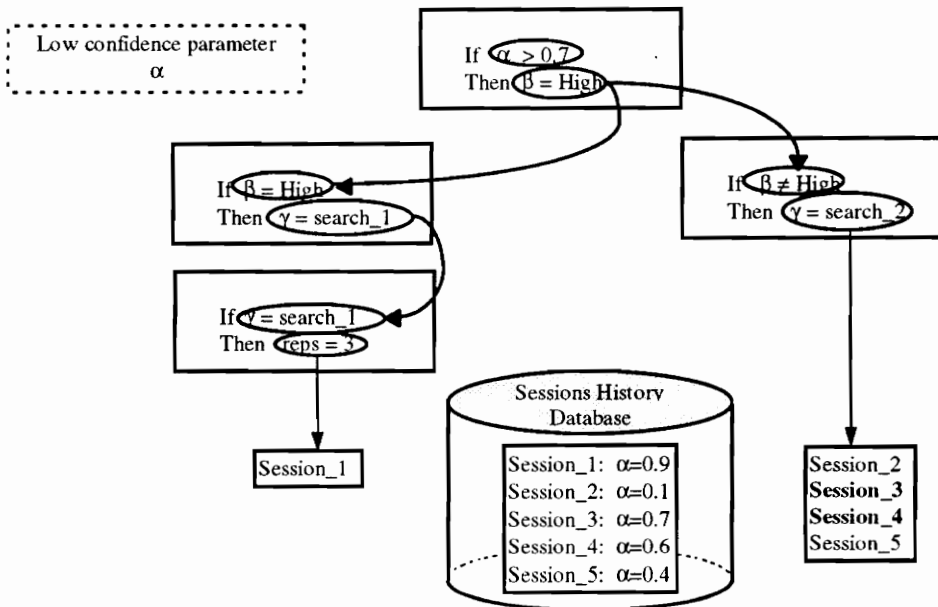


Figure 5.10. Details of the Domain Knowledge component of the Discovery Learner



5.11a. Initial Link-of-Influence



5.11b. Subsequent Link-of-Influence

Figure 5.11. Link-of-Influence

The final submodule currently present in the domain-knowledge portion of the Learner is the classifier-methodology-to-search-space (CM/SS) mapping. Recall that the Classifier KBSOS synthesizes simulation runs and then characterizes the resulting optimization surface according to six output measures. These output measures have been chosen particularly because they channel surfaces toward the search technique most appropriate for the type of surface.

The purpose of the CM/SS component as used in the Discovery Methods module is to suggest new search strategies for appropriate bad and ugly cases. Three current means of doing this in the CM/SS include what we call the "primitive method," whereby Smith's (1973) second and third search choices in his experiments are suggested; a taboo-region method, where those strategies deemed terrible in a particular region of classifier methodology/search space are listed as "to be avoided"; and a third method that calculates the Mahalanobis distance from the currently recommended strategy to the nearest centroid of the other strategies. As noted in Figure 5.10, the CM/SS rules impact only the Strategy Selector module in the KBSOS.

Discovery Methods module

The final Learner module to be discussed is the "work-horse" component, namely the Discovery Methods module. Recall that, as Frawley et al.'s (1992) paradigm suggests, discovery methods consist of search followed by evaluation. The search itself, they say, also has two parts: pattern identification and concept description. As mentioned, the former defines classes that maximize within-class similarity while minimizing among-class similarity. Concept description consists of deriving descriptions of the classes.

Our discovery method module also consists of search and evaluation, the latter of which we have labeled our "experimental designer" (in the sense of a "design-of-experiments" expert). The pattern identification phase of our search consists of the four tasks, parameter modification, specialization, rule modification,

and generalization. The first three tasks are defined procedurally in Crouch (1992), and generalization is described in Greenwood et al. (1993). The procedures referenced are modified as explained in the example below. These four tasks are conducted instead of a more formal cluster analysis, although, in a sense, most of the four tasks pursue their goal through attempts at clustering BUG cases into clearer categories. The second portion of search, the concept description effort, utilizes rules as the representation scheme in which all new constructs will be expressed. This is both convenient, given that the four tasks are designed to operate on rules; and propitious for further discovery, since any rule in the KBSOS or Learner can, whether a new or an old construct, in principle, then be re-learned (i.e., modified, or even "unlearned," etc.) by additional search using the four tasks.

Further particular details of the search and experimental design modules will not be discussed in this section. Rather, an example implemented in practice will now be discussed that illustrate discovery learning through parameter modification.

BUILDING A SYSTEM: A PARAMETER MODIFICATION EXAMPLE

This section will show an example of parameter modification. The system will modify the threshold of the parameter coefficient of variation (CV) in an attempt to obtain "better" results. The example is based on multiple simulation studies actually run previously by us and stored in the sessions base. The rules

used in these sessions have been preserved, and relevant portions are shown below as needed in tabular form.

Presented in summary fashion, the example proceeds as follows. First the low-confidence parameter list provides the needed bias (Figure 5.7) in the form of those parameters that most need to be checked; we assume that the parameter CV is at the top of the list. The link-of-influence (LOI) module, next determines those sessions potentially affected by the suggested change. Subsequently the range-of-parameter-sensitivity module performs a line search to suggest needed simulation runs (i.e., the “experimental points”), while the experimental designer (the evaluation module of the discovery methods) runs the affected sessions and evaluates performance. The details of the parameter modification example are presented below.

At the top of the low-confidence parameter list is the parameter coefficient of variation (CV). The Learner passes this parameter to the LOI, which proceeds to track the affected rules by forward chaining.

The classifier is the first knowledge base that is activated in the KBSOS, so forward chaining starts there. In general, to forward chain through the knowledge base, the antecedents of the rules must be checked for matches with the parameter CV. If the parameter in the antecedent of a rule matches, the variable in the consequent is also placed on a list (call it “list1”). Therefore, if after checking all the rules for matches there is at least one match, then the process of checking and matching proceeds, but with the consequents of the rules in list1. When there are no more matches, this phase terminates, and the next knowledge base is checked using all of the parameters listed to this point.

Table 5.1 contains relevant portions of the Classifier knowledge base, with each row representing one rule. That table contains three columns: the first contains the rule name, the second the rule’s antecedent, and the third its consequent. As can be seen in Table 5.1, two rules (named var2 and var3) contain CV in the antecedent. So the consequents of rules var2 and var3, namely “random_error,” are

placed on list1. Since items were added to list1, another round of checking needs to be done, but this time the effort is to match antecedents with the parameter random_error. Table 5.1 shows two rules (opt_dist1, opt_dist2) that have random_error in the antecedents; therefore, the consequent "dist_to_opt" in the rules opt_dist1 and opt_dist2 are added to list1. Again, items are added to list1, so another round of matching follows, but this time using the parameter dist_to_opt to match with the antecedents of the rules. At this point there are no more matches, so this phase can terminate. The parameters on list1, CV, random_error, and dist_to_opt, are passed on to the next phase.

Table 5.1. A portion of the Classifier Knowledge Base.

Name	Antecedent	Consequent
loc_opt1	synth = done	CALL COMBINSS,"" CALL RANK,"" l_opt_thresh = 0.15 SHIP locopt, l_opt_thresh CALL OPTIMA,"" RECEIVE opttot, num_opt opt = done;
loc_opt2	num_opt = 1	local_optima = absent;
loc_opt3	num_opt > 1	local_optima = present;
var1	synth = done	RECEIVE avevar, variance cvar = done;
var2	cvar = done AND CV < 0.5	random_error = small;
var3	cvar = done AND CV ≥ 0.5	random_error = large;
opt_dist1	local_optima = absent AND random_error = small AND last_search = RSM	dist_to_opt = near;
opt_dist2	local_optima = present OR random_error = large	dist_to_opt = far;
fac_act1	synth = done	CALL factact,"" RECEIVE factive, active act=done;
fac_act2	active < (0.5*num_inputs)	factor_activity = low;
fac_act3	active >= (0.5*num_inputs)	factor_activity = high;

The Selector is the next knowledge base activated in typical KBSOS operation so it is searched next. To forward chain, antecedents of the Selector rules are checked for matches with the parameters CV, random_error, and dist_to_opt. If any rule has an antecedent that matches any of these parameters, the variable in the consequent of the rule is placed on another list, called list2. If after checking all the rules for matches there is at least one item on list2, then the process of checking and matching repeats with all items on list2. When there are no more matches, this phase terminates, and the next knowledge base incurred is checked using all of the parameters listed to this point.

As can be seen in Table 5.2 the rule simplex_1 has matches on random_error and dist_to_opt. So the consequent of the rule is added to list2, namely the variable "search_strategy." Since search_strategy does not match any other antecedents in Table 5.2, this phase terminates, but passes on the variables on list2, namely CV, random_error, dist_to_opt, and search_strategy.

Table 5.2. A portion of the Selector Knowledge Base

Name	Antecedent	Consequent
random_1	num_factors = small AND num_sim_runs = petite	search_strategy= random_search;
full_factorial_1	num_factors = small AND num_sim_runs = large AND local_optima = absent AND random_error = large AND factor_activity = high	search_strategy = RSM_II;
Simplex_1	num_sim_runs = medium AND local_optima = absent AND dist_to_opt = near AND random_error = small AND factor_activity = high	search_strategy = RSM_I;

The next knowledge base to be activated is the Detailer. (See Table 5.3.) To forward chain through that knowledge base, the antecedents of the Detailer rules are checked for matches with the items CV, random_error, dist_to_opt and search_strategy. Table 5.3 shows that there are three rules (rand_srch, simplex, full_factorial) that match the variable search_strategy in the list passed from the previous phase.

No other parameters are a match. So the rules `rand_srch`, `simplex`, and `full_factorial` are put on a new list, `list3`. Since there were matches another round of matching follows, but now using the consequent variables `starting_point`, `region_of_fit`, `step_size`, and `design`. There are no matches so this phase and the link-of-influence module terminates. In summary so far, the items in the lists constitute the link-of-influence, which indicates the potential impact of a change in the parameter under consideration for modification. A change in CV can affect the factors `random_error`, `dist_to_opt`, `search_strategy`, `starting_point`, `region_of_fit`, `step_size`, and `design`.

Table 5.3. A portion of the Detailer Knowledge Base

Name	Antecedent	Consequent
<code>rand_srch</code>	<code>search_strategy=random_search</code>	<code>starting_point=none</code> <code>CALL REGION,""</code> <code>region_of_fit = found</code> <code>step_size = none</code> <code>design = none;</code>
<code>simplex</code>	<code>search_strategy = RSM_I</code>	<code>starting_point = start</code> <code>CALL REGION,""</code> <code>step_size = (0.10 * (ub - lb))</code> <code>design = simplex;</code>
<code>full_factorial</code>	<code>search_strategy = RSM_II</code>	<code>starting_point = start</code> <code>CALL REGION,""</code> <code>step_size = (0.10 * (ub - lb))</code> <code>design = full_factorial;</code>

Note that if the parameter to be modified had been in the consequent of a rule rather than in its antecedent, then the scanning process would have proceeded as above except that scanning would have been done on consequents and antecedents would have been posted to the lists. This is analogous to backchaining.

Note that the primary interest at this point is in finding sessions that are affected by the change in the parameter threshold. In the Detailer KBSOS there are only five variables that can affect a session: (1)

search_strategy, (2) region_of_fit, (3) starting point, (4) step-size, and (5) design. If none of these factors shows up on the list generated above of affected parameters, then there is no reason to pursue parameter modification any further; if, however, at least one of the previously mentioned five factors is affected, then the range-of-parameter-sensitivity module must be called. Since all five factors are affected, further effort is necessary. Continuing to follow the same logic as illustrated in Figure 5.11, the Sessions Base is examined to see which of the many sessions there are affected by a change in CV. Five sessions are identified, and we call them session 1 to session 5 here for simplicity.

Some of the information stored in the Sessions Base regarding these sessions is included in Table 5.4. Note that each row entry in that table consists of an entire simulation optimization session actually conducted in the past. As can be seen in Table 5.4, past sessions were conducted using different model conditions (e.g., different demand distributions), each taking a different number of runs to reach optimality, which itself varied from case to case. Note that all optimizations in Table 5.4 were performed using a “Simplex” experimental design; this is not surprising because all CV’s are below $CV = 0.05$. Table 5.1, rule var2, indicates that consequently random_error = small; Table 5.2 , rule simplex_1, informs that search_strategy = RSM_I, which is the Simplex search. Also note from Table 5.4 that the range-of-parameter-sensitivity = (0.0021, 0.0216)

Table 5.4. Some Sessions Base Data

Session Name	Demand Distribution	Search Strategy	Response CV	Optimal Cost/Day	Runs Made
1	Unif(0.19, 0.21)	Simplex	0.0021	\$2.82	65
2	Exp(0.20)	Simplex	0.0085	\$2.86	55
3	Gamma(0.25, 0.80)	Simplex	0.0105	\$2.81	39
4	Gamma(0.1111, 1.80)	Simplex	0.0200	\$3.79	28
5	Gamma(0.05, 4.0)	Simplex	0.0216	\$4.08	30

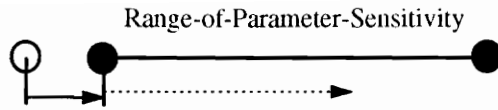
The experimental-design module is invoked next to perform a line search using the value of the parameter taken from the low-confidence parameter list ($CV = 0.5$) and the range-of-parameter-sensitivity (0.0021,

0.0216). Note that the situation we have is that of Figure 5.12b; we will be examining parameter changes “from above.” This situation is redrawn in Figure 5.12d, where the CVs of the five sessions are shown explicitly within the range-of-parameter-sensitivity.

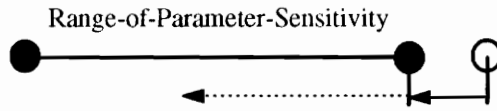
The experimental designer first tries moving the parameter $CV = 0.5$ to that of session 5. It does this by scheduling an entire simulation optimization under the same conditions as before, except that now rule var2 in Table 5.1 will be $IF\ CV < 0.0216$. This results in the search strategy shifting from RSM_I (Simplex) to RSM_II (Full Factorial).

The results obtained constitute an improvement in that there is a tie on cost (i.e., costs are within 10% of each other), but the number of runs is reduced from 30 to 25.

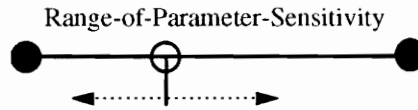
Since session 5 resulted in an improvement with the change in CV from 0.5 to 0.0216, a further shift in CV from 0.0216 to 0.0200 (session 4) is considered. Therefore, another entire simulation optimization is conducted on session 4’s condition, this time using a Full Factorial design. As there is a significant improvement in cost from \$3.79/day to \$2.69/day (a 41% improvement), another shift in CV, to that of session 3 is considered. Since performance deteriorates (tie on cost, but runs more costly), the process terminates. As the recommendation from the Discovery Learner is to change the CV, rules var2 and var3 in the classifier knowledge base are modified. The factor CV is left on the Low-Confidence Parameter List for further investigation by the Learner after new sessions are accumulated.



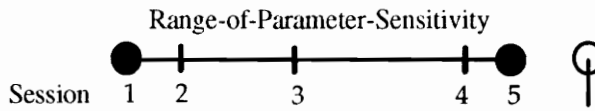
5.12a. From below



5.12b. From above



5.12c. From within



5.12d. The case involving the five relevant sessions in the Sessions Base.

Figure 5.12. Examples of Line Search

CONCLUSIONS AND FUTURE WORK

Previous research has emphasized the advantages of building a knowledge-based simulation optimization system and of the potential for an embedded learner. Whereas earlier work describes the architecture of such a learner, this research reports on a learner (or portions thereof) we have actually built and its experiences in adapting itself to a history of simulation runs of inventory problems. The Learner itself was changed to include the concept of discovery learning, whereby the optimizer develops its own agenda of problems to pursue.

The example described in this research successfully experimented in changing a parameter in the knowledge base of the simulation optimizer, one of four types of learning (parameter modification) described in the literature. Although the other three types of learning (specialization, rule modification, and generalization) are in most ways easier than the learning shown here, it is not recommended that these types of learning be pursued next. Rather it is believed that a study be made of incorporating domain knowledge to see if the number of confounding factors in the experimental-design portion of the Learner can be reduced.

What is meant by reducing the confounding factors is that there are often too many factors involved in each simulation optimization session to identify which are important. For example, there is starting point, inter-gridpoint spacing, and step size as well as CV, search method, etc. For the experimental designer to hope to be able to attribute change to the proper factor, the other influences must be properly controlled. This will result in huge experimental designs unless domain knowledge can be brought to bear to eliminate possible confounding factors or to rule out the wisdom of testing on a factor by factor basis. Further work is needed to identify more specifically which factors can be learned practically speaking, and how domain knowledge can be included to reduce the complexity of the task. We believe this is the next

critical topic which must be pursued in the development of a Learner for a knowledge-based simulation optimization system.

Chapter Six: Conclusions

Surface Study

In Chapter three a simple, inventory-simulation model was studied under four different experimental design conditions. These conditions varied the coefficient of variance of demand and of lead time and also examined two different levels of design conditions, i.e., the number of replications and the simulation run length. A simple model was studied because it was believed that even a naive modeler intent on finding the system optimum would be able to safely and properly use a technique such as RSM, a widely used and respected approach.

The purpose of the study was to investigate common statistical measures over the search region. Both point estimates (mean, standard deviation, coefficient of variation, signal-to-noise ratio) and region measures and tests (normality of residuals, homogeneity of variance, significance-of-regression and lack-of-fit) were examined.

Point-estimate measures exhibited considerable sensitivity to experimental design conditions. This gave rise to concerns that perhaps the simple inventory model might not be simple enough to conduct simulation-optimization searches using methods requiring some parametric statistical tests. Regional measures added some additional concerns.

That the appropriateness of various optimization approaches should be questioned was portrayed in a final set of plots indicating which points of the overall search area were amenable to first-order RSM and which were not. It was found that an important determinant of amenability was the inter-gridpoint spacing of the gridpoints. The gridpoint spacing is a very important practical issue, as one conducting optimization on a simulation model must be able to specify, e.g., in RSM, the (uncoded) size of the region of the first-order designs and the step size to be taken along the path of steepest ascent/descent. It was found for a spacing of $\Delta = 40$ that in no case were more than 10% of the total number of regions appropriate for first-order RSM. For $\Delta = 20$, the range of appropriate percentages varied from about 25% to 78%.

Again, this is a relevant, practical finding. Individuals conducting optimization must be very careful not to make experimental-region size too large, since then first-order parametric metamodels may only be appropriate 10% of the time, whereas setting even a smaller region size will still lead to considerable variability in achieving a properly executed search.

There are three implications of these findings. The first is that there is a need to develop a simulation-optimization “pre-processor” or “starter” that suggests both a starting point for the optimization and the granularity of the problem, i.e., the inter-gridpoint spacing or some surrogate (chapter four addresses this). Many times it is appropriate to assume that a “good-enough” starting point is known by an expert, but even if so, it is not as clear that such an expert would have sufficient knowledge to specify an inter-gridpoint spacing that is not too big, given the particular model variabilities (exogenous and endogenous) and design conditions (run length and replications). Too small a spacing may be costly.

The second implication of the findings of this research is that nonparametric metamodeling should be examined. This is necessary not only because of the potential of violating parametric assumptions, but also for another reason implied in this research: the benefits of global, nonparametric metamodeling. Recall that in Figure 3.5 with two replications, multi-modal response surfaces were indicated (which was incorrect). If RSM were attempted starting on the wrong “side” of such a simulation response surface, the wrong optimum might be found. A possible alternative to parametric metamodeling such as RSM is global nonparametric metamodeling, whereby the whole surface is modeled using a nonparametric technique such as kernel smoothing or spline smoothing. In fact, some preliminary investigation (Keys, Rees, Greenwood, (1995)) suggests that global, nonparametric metamodeling is very effective, seems safer, and requires relatively few computer runs to obtain the optimum.

The third implication of this research is that a multi-strategy approach to simulation optimization be explored. Since a response surface may vary considerably over the entire region in terms of both point and region characteristics/measures, it stands to reason that different search techniques might be appropriate and thus more successful in different areas of the search space. For example, RSM might be appropriate in one area and random search in another. Initial studies on this have already been done (Crouch, Greenwood, Rees, (1995), Greenwood, Rees, Crouch (1993)).

Best First Search Approach

Chapter four defined a Starter for use in those simulation optimization cases where either the starting point or the granularity of the problem are not known in advance. The Starter combines the artificial-intelligence based best-first search with a divide-and-conquer strategy and a safety net.

Three examples have illustrated the Starter procedure. The first example showed that the Starter worked on a “simple” simulation-optimization problem, while the second illustrated the process on a more involved surface with twenty times the variance of the first. The final example represented a very difficult surface to optimize, with multimodal behavior and large flat regions. The Starter worked quite well even without the safety net in all cases, and obtained an optimal solution within 7% after one pass of the safety net in all three cases.

A lower-bound estimate of the number of simulation runs N required by the Starter may be determined based on the original user-specified region and inter-gridpoint minimal spacing. If a best case of the minimal subdivisions in the divide-and-conquer step occurs, and if no ties occur among the most preferred region at any point in the algorithm and the other regions, then for a square region with a granularity Δ_u ,

$$N = 12 + 15m$$

$$\text{where } m = \log_2 \left\lceil \frac{\text{range}}{\Delta_u} \right\rceil \text{ and } \lceil \bullet \rceil \text{ is the ceiling function.}$$

For example, if $\Delta_u = 11$ and the user specifies $48 \leq x \leq 400$, then

$$\text{range} = 400 - 48 = 352$$

$$m = \log_2 \left\lceil \frac{352}{11} \right\rceil = \log_2 \lceil 32 \rceil = 5, \text{ and}$$

N= 87 simulation runs.

Conversely, the maximum number of runs required over the specified range to completely cover a square feasible region at a granularity of Δ_u if the Starter approach is ignored is

$$N = 3 \left[\left(\frac{\text{range}}{\Delta_u} \right) + 1 \right]^2$$

In the example above,

$$N = 3 \left[\left(\frac{352}{11} \right) + 1 \right]^2 = 3,267 \text{ simulation runs.}$$

The potential savings using the Starter algorithm is great.

Future work on the Starter involves testing it on more surfaces. The results from these runs can be used to ascertain how aggressiveness may be incorporated dynamically into the Starter process and also to determine which metamodel should be used in estimating the performance of each region, as discussed in chapter four.

Building A KBSOS With Discovery Learning

Previous research has emphasized the advantages of building a knowledge-based simulation optimization system and of the potential for an embedded learner. Whereas earlier work describes the architecture of such a learner, this research reports on a learner (or portions thereof) we have actually built and its experiences in adapting itself to a history of simulation runs of inventory problems. The Learner itself was changed to include the concept of discovery learning, whereby the optimizer develops its own agenda of problems to pursue.

The example described in this research successfully experimented in changing a parameter in the knowledge base of the simulation optimizer, one of four types of learning (parameter modification) described in the literature. Although the other three types of learning (specialization, rule modification, and generalization) are in most ways easier than the learning shown here, it is not recommended that these types of learning be pursued next. Rather it is believed that a study be made of incorporating domain knowledge to see if the number of confounding factors in the experimental-design portion of the Learner can be reduced.

What is meant by reducing the confounding factors is that there are often too many factors involved in each simulation optimization session to identify which are important. For example, there is starting point, inter-gridpoint spacing, and step size as well as CV, search method, etc. For the experimental designer to hope to be able to attribute change to the proper factor, the other influences must be properly controlled. This will result in huge experimental designs unless domain knowledge can be brought to bear to eliminate possible confounding factors or to rule out the wisdom of testing on a factor by factor basis. Further work is needed to identify more specifically which factors can be learned practically speaking, and how domain knowledge can be included to reduce the complexity of the task. We believe this is the next

critical topic which must be pursued in the development of a Learner for a knowledge-based simulation optimization system.

BIBLIOGRAPHY

- Azadivar, F.(1992), "A Tutorial on Simulation Optimization," *Proceedings of the Winter Simulation Conference*, pp. 198-204.
- Barton, R.R.(1992), "Metamodels for Simulation Input-Output Relations," *Proceedings of the Winter Simulation Conference*, pp. 289-299.
- Box, G.E.P., and K.B. Wilson(1951), "On the Experimental Attainment of Optimum Conditions," *Journal of the Royal Statistical Society, Ser. B*, 13, 1, pp. 1-45.
- Carbonell, J.G., R.S. Michalski, and T.M. Mitchell (1983) "An overview of machine learning," *Machine Learning: An Artificial Intelligence Approach*, Michalski, Carbonell and Mitchell, eds., Palo Alto, CA: Tioga Publishing Company.
- Conover, W.J., M.E. Johnson, and M.M. Johnson (1981), "A Comparative Study of Tests for Homogeneity of Variances, with Applications to the Outer Continental Shelf Bidding Data," *Technometrics*, 23, 4, pp. 351-361.
- Conte, S.D., and C. de Boor(1980), *Elementary Numerical Analysis: An Algorithmic Approach*, McGraw-Hill, New York.
- Crouch, I.W.M.(1992), Dissertation, *A Knowledge-Based Simulation Optimization System with Machine Learning*, Department of Management Science, Virginia Polytechnic Institute & State University, Blacksburg, VA.
- Crouch, I.W.M., A.G. Greenwood, and L.P. Rees(1995), "Use of a Classifier in a Knowledge-Based Simulation Optimization System," *Naval Research Logistics*, to appear.
- D'Agostino, R.B., and M.A. Stephens(1986), eds., *Goodness-Of-Fit Techniques*, Marcell Dekker Inc., New York.
- Doran, J.E. and D. Michie(1966), "Experiments with the Graph Traverser Program," *Proceedings of the Royal Society of London, Ser. A*, 294, pp. 235-259.
- Encyclopedia of Artificial Intelligence*(1987), S. Shapiro ed., John Wiley & Sons, NY.

- Frawley, W.J., G. Piatetsky-Shapiro, and C.J. Matheus (1992), "Knowledge Discovery in Databases: An Overview," *AI Magazine*, 13(3): pp. 57-70.
- Glaser, R.E.(1983), "Levene's Robust Test for Homogeneity of Variance," *Encyclopedia of Statistical Sciences*, 4, pp. 608-610.
- Good, I.J., R.A. Gaskin(1971), "Nonparametric Roughness Penalties for Probability Densities," *Biometrika*, 58, 2, pp. 255-277.
- Greenwood, A.G., L.P. Rees, and I.W.M. Crouch(1993), "Separating the Art and Science of Simulation Optimization: a Knowledge-Based Architecture Providing for Machine Learning," *IIE Transactions*, 25(6), pp. 70-83.
- Holland, J.H., K.J. Holyoak, R.E. Nisbett, and P.R. Thagud(1986), *Induction: Processes of Inference, Learning, and Discovery*, Cambridge, MA: MIT Press.
- Jacobson, S.H., and L.W. Schruben(1989), "Techniques for Simulation Response Optimization," *Operations Research Letters*, 8, 1, pp. 1-9.
- Keys, A.C., L.P. Rees, and A.G. Greenwood(1995), "A Performance-Based Study of Nonparametric-Metamodeling Techniques for the Classification of Response Surfaces in Simulation Optimization," working paper.
- Keys, A.C., L.P. Rees, and A.G. Greenwood(1995), "Implementing Nonparametric-Metamodeling Techniques in a Classifier Knowledge-Based Simulation Optimization System," working paper.
- Klimasauskas, C., J. Guiver, and G. Pelton (1989), *NeuralWorks Professional II and NeuralWorks Explorer*, Pittsburgh, PA: NeuralWare, Inc.
- Müller, H.(1984), "Optimal Designs for Nonparametric Regression," *Statistics and Probability Letters*, 2, pp. 285-290.
- Mathewson, S.C.(1984), "The Application of Program Generator Software and Its Extensions to Discrete Event Simulation Modeling," *IIE Transactions*, 16, pp. 3-18.
- Meketon, M.S.(1987), "Optimization in Simulation: A Survey of Recent Results," *Proceedings: Winter Simulation Conference*, pp. 58-67.
- Michalski, R.S (1986), "Understanding the nature of learning: issues and research directions," *Machine Learning: An Artificial Intelligence Approach*, Michalski, Carbonell and Mitchell, eds., Los Altos, CA: Morgan Kaufman Publishers, Inc.
- Myers, R.H.(1971), *Response Surface Methodology*, Allyn and Bacon, Boston.

- Neter, J., W. Wasserman, and M.H. Kutner(1985), *Applied Linear Statistical Models*, (2nd edition), Irwin, Illinois, pp. 618-622.
- O'Keefe, R.M.(1986), "Advisory Systems in Simulation," in E. J. H. Kerckhoffs, G. C. Vansteenkiste, and B. P. Zeigler (Eds.) *Artificial Intelligence Applied to Simulation, The Society for Computer Simulation*, San Diego, CA, pp. 73-78.
- Pritsker, A.A., and P.J. Kiviat(1969), *Simulation With GASP II*, Prentice-Hall, Englewood Cliffs, NJ.
- Pritsker, A.A., and C.E. Sigal(1983), *Management Decision Making--a Network Simulation Approach*, Prentice-Hall, Englewood Cliffs, NJ.
- Quinlan, J.R. (1986), "Induction of Decision Trees," *Machine Learning*, 1(1), pp. 81-106.
- Quinlan, J.R.(1979), "Discovering Rules by Induction from Large Collections of Samples," in D. Michie (Ed.), *Expert Systems in the Microelectronic Age*, Edinburgh University Press, pp. 168-201.
- Rumelhart, D.E. and J.L. McClelland(1986), *Parallel Distributed Processing*, vol. 1, Cambridge, MA: MIT Press.
- Safizadeh, M.H. (1990), "Optimization in Simulation: Current Issues and the Future Outlook," *Naval Research Logistics*, 37, pp. 807-825.
- Simon, H.A. (1983), "Why should machines learn?" *Machine Learning: An Artificial Intelligence Approach*, Michalski, Carbonell and Mitchell, ed., Palo Alto, CA: Tioga Publishing Company.
- Slade, S.(1991), "Case-based Reasoning: A Research Paradigm," *AI Magazine*, 12(1), pp. 42-55.
- Siochi, F.C. (1993) Dissertation Proposal, *Building a Knowledge-Based Simulation Optimization System with Discovery Learning*, Department of Management Science, Virginia Polytechnic Institute & State University, Blacksburg, VA.
- Smith, D.E.(1973), "An Empirical Investigation of Optimum-seeking In the Computer Simulation Situation," *Operations Research*, 21(2), pp. 475-497.
- Stanwood, K.L., L.N. Waller, and G.C. Marr(1986), "System Iconic Modeling Facility," *Proceedings of the Winter Simulation Conference*, pp. 531-536.
- Tocher, K.D.(1966), "Some Techniques of Model Building," *Proceedings of the IBM Scientific Computing Symposium on Simulation Models and Gaming*, IBM, White Plains, NY, pp. 119-155.
- Tocher, K.D.(1962), *The Art of Simulation*, English Universities Press, London.

Toothaker, L.E.(1991), *Multiple Comparisons for Researchers*, Sage Publications, Inc., Newbury Park, CA.

Wilson, J.R.(1987), "Future Directions in Response Surface Methodology for Simulation," *Proceedings: Winter Simulation Conference*, pp. 378-381.

Winston, P.H.(1984), *Artificial Intelligence*, Addison Wesley, Reading, MA.

APPENDIX A

In the simulation used in Examples 1 and 2 of chapter three, the implicit time unit is assumed to be one day. Demand for both examples comes from a gamma distribution, $\Gamma(\alpha, \beta)$, where α is the shape parameter and β is the scale parameter. Demand for both examples has a mean of 0.2 days. Example 1, however, has a shape parameter of 1.0, indicating an exponential distribution, whereas Example 2 has an α of 0.05. Example 1 also has a deterministic lead time of zero, whereas Example 2 has a lead time following a truncated normal distribution. The simulation is designed to start at the beginning of a cycle and to terminate on the completion of a cycle. A cycle (T) is defined to exist from the moment just after an order arrives until the time the next order arrives (see Figure 3.1).

The simulation model has four main events:

- **Initialization**
- **Arrival-of-Demand**
- **Arrival-of-Order**
- **Termination**

Initialization specifies the values of several system parameters and variables. A sequence of arrival-of-demand and arrival-of-order events follows initialization, thereby starting the actual simulation. The simulation consists of a warm-up period followed by a study period. After the warm-up period all

statistics are reset to zero, and statistics collection for the study period begins. At termination, the performance measure is calculated.

The four main events can be broken down into activities.

The **Initialization event** has the following activities:

- **Set decision variables Q and R (They do not change during the simulation run.)**
- **Set Inventory and Inventory Position to Q**
- **Set Number of Back Orders and Number of Orders to zero**
- **Set the minimum time for warm-up (250 days) and statistics collection (1000 days)**

The **Arrival-of-Demand** event has two activities:

- **Check Order**
If Inventory Position is less than or equal to R then
 Place Order
 Increment Inventory position by Q
 Schedule Arrival-of-order
- **Process Demand**
If demand can be met from inventory then
 decrement Inventory and Inventory Position
If demand cannot be met then
 Satisfy whatever can be satisfied from Inventory
 Set number of backorders to the portion of demand that cannot be satisfied
 If this is the first time in the cycle that Inventory cannot satisfy demand then
 Set Inventory and Inventory Position to zero (0)
 If this is not the first time in the cycle that Inventory cannot satisfy demand then
 Subtract demand from Inventory Position

The **Arrival-of-Order** event has three activities:

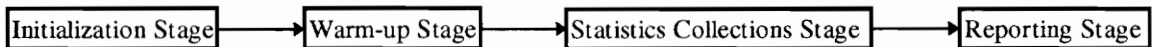
- **Process Order**
Satisfy backorders as possible
Increment Inventory Position by Q
If Inventory Position is greater than zero then
 Set Inventory to Inventory Position

- **Check Order**
 If Inventory Position is less than or equal to R then
 Place Order
 Increment Inventory position by Q
 Schedule Arrival-of-order
- **Check-for-End-of-Cycle**
 If current time is \geq warm-up time (250) and it is still the warm-up period then
 Set the period to statistics collection
 Set simulation termination time to current time plus statistics collection time (1000)
 Set total warm-up time to current time
 Set number of orders to zero (0)
 Start statistics collection
 If current time \geq simulation termination time then
 Terminate Simulation

The **Termination** event has one activity

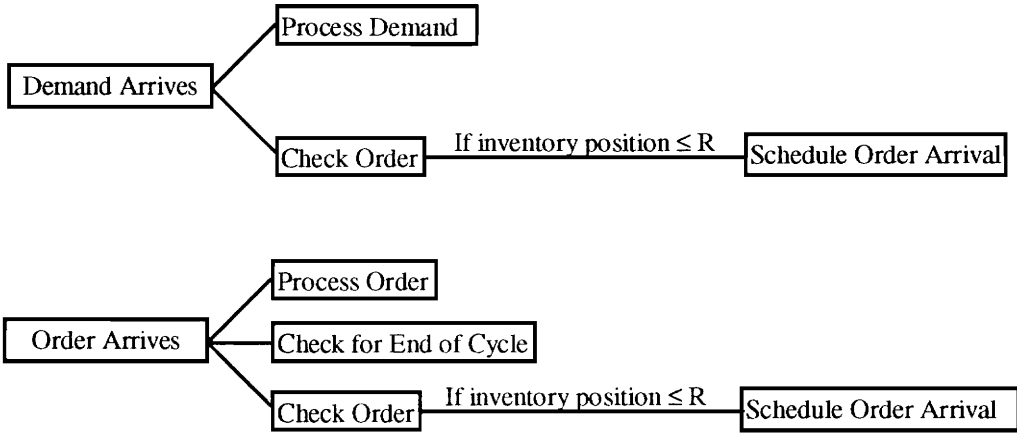
- **Generate Report**
 Calculate Statistics
 Average Daily Inventory
 Average Daily Backorder
 Total number of orders
 Total Cost for run normalized to Average Daily Cost
 Print Q, R, Average Daily Cost

The four events occur during four separate stages, an initialization stage, a warm-up stage, a statistics-collection (study) stage, and a reporting stage. The four stages are sequential as shown in the following diagram.

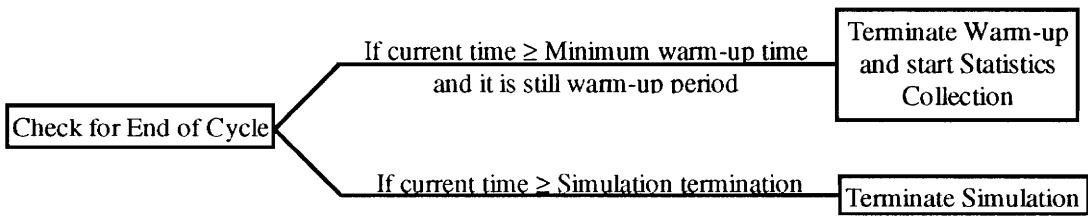


The initialization stage corresponds to the initialization event. Both the warm-up stage and statistics-collection stage are a series of arrival-of-demand and arrival-of-order events. The reporting stage corresponds to the termination event.

The following diagrams show how the arrival-of-demand and arrival-of-order events interact in the warm-up stage.



The statistics collection stage has the same basic events. The check-for-end-of-cycle activity is shown in greater detail below showing the termination of warm-up and the beginning of statistics collection. The difference between the warm-up and the statistics-collection stages is that the statistics that are collected in the warm-up stage are discarded.



The reporting stage essentially starts at the end of the simulation. The measure of performance is the total cost of the simulation run normalized to a daily cost. Total cost is composed of holding cost, backorder cost, and order cost. The holding cost and backorder cost are based on inventory and backorder levels, which are maintained by the simulation program as time-persistent variables. The daily order cost for the duration of the statistics-collection stage is calculated as the cost per order times the number of orders placed during the study period, divided by the duration of the statistics-collection stage (expressed in

days). Summing the holding cost, backorder cost, and order cost gives the daily cost for the simulation run. The output of the reporting stage is the (Q, R) pair and its associated daily cost.

APPENDIX B

Pseudocode for the best-search algorithm of chapter four is as follows:

MAIN

User

The user specifies the dimensionality (k) of the simulation-optimization problem.
The user specifies whether optimization is minimization or maximization.
The user specifies the region to be optimized.
The user specifies the minimum inter-gridpoint spacing, $\Delta_{\mathbf{u}}$.

Initialize

If the user-specified region is not convex, create the minimum number of polygons (each of which is convex, by definition).
Define each polygon as a “promising region” or as a “region,” for short.
Initially define \mathcal{E} = list of promising regions to be explored = $\{\emptyset\}$.
Initially define \mathcal{D} = list of regions to be discarded = $\{\emptyset\}$.
Define each vertex of each polygon as a “gridpoint.”
Define Δ_i as the inter-gridpoint spacing along dimension i .
If unspecified by user above, set $\Delta_i = 3.125\%$ of initial range in direction i
(Halving 100% five times (50; 25; 12.5; 6.25; 3.125) so that the search is limited to five levels).
Run replications (e.g., 3 - depending on aggressiveness) at each gridpoint of all regions.
Place all regions on the list \mathcal{E} .
SORT the list \mathcal{E} , with the most preferred region on the front of the list.

Best-first search with safety net

```
Repeat
  While the list  $\mathcal{E}$  is not empty
    Take the first region  $\mathcal{R}$  off list  $\mathcal{E}$ .
    Initialize  $\text{stop\_criteria}(\mathcal{R}) = \text{false}$ .
    While not( $\text{stop\_criteria}(\mathcal{R})$ )
      SUBDIVIDE region  $\mathcal{R}$ 
      Perform multiple comparison test on list  $\mathcal{E}$ 
      Sort list  $\mathcal{E}$  with the most promising region at the front of the list
    End while
  End while
  SORT list  $\mathcal{D}$  with the most promising region at the front of the list
  While the list  $\mathcal{D}$  is not empty
    Perform SAFETY NET on list  $\mathcal{D}$ .
  End while
Until list  $\mathcal{E}$  and list  $\mathcal{D}$  are empty
```

STOP

END MAIN

LOGICAL FUNCTION STOP_CRITERIA(\mathcal{R})

```
/* User specified stopping criterion */
If user-supplied  $\Delta_U$  is reached then
  stop_criteria = true
  annotate the region  $\mathcal{R}$  as having reached  $\Delta_U$ .
  RETURN
endif.
If assumptions_are_met( $\mathcal{R}$ ) then
  /* gradient search looks promising */
  If  $F_{\text{REGR}}$  is significant and  $F_{\text{LOF}}$  is not significant for this  $\mathcal{R}$  then
    stop_criteria = true
    annotate the region  $\mathcal{R}$  as being good for RSM.
    RETURN
  endif.
  /* Region is flat */
  If  $F_{\text{REGR}}$  and  $F_{\text{LOF}}$  are both not significant for this  $\mathcal{R}$  then
    stop_criteria = true
    annotate the region  $\mathcal{R}$  as not being good for RSM.
    RETURN
  endif.
endif.
```

RETURN

END STOP_CRITERIA(\mathcal{R})

LOGICAL FUNCTION ASSUMPTIONS_ARE_MET(\mathcal{R})

```
/* If errors are not normal */
If Shapiro-Wilk test is significant then
    assumptions_are_met = false;
else
    /* If errors are normal but do not have homogeneous variance */
    if Bartlett-Box test is significant then
        assumptions_are_met = false;
    else
        /* If errors are normal and have homogeneous variance */
        assumptions_are_met = true.
```

RETURN
END ASSUMPTIONS_ARE_MET(\mathcal{R})

PROCEDURE SUBDIVIDE(\mathcal{R})

```
Define the list  $\mathcal{T}$  as a temporary list.
/* Take the parent off the list */
Remove  $\mathcal{R}$  from list  $\mathcal{E}$ 

For each dimension  $\iota$  in the problem space /*  $1 \leq \iota \leq k$  */
    bisect each region  $\mathcal{R}$  boundary along the  $\iota$ th axis
    create a new gridpoint at each bisected boundary if one does not already exist
Next  $\iota$ .

Without straying outside the region,
    form the Cartesian product of each new gridpoint, thereby creating more new gridpoints.
Run replications (e.g., 3) at each of the new gridpoints that do not already have runs.
Create up to  $2^k$  new, non-overlapping, smaller regions completely covering  $\mathcal{R}$ .
Place each new region on the list  $\mathcal{T}$ .
Perform MULTIPLE_COMPARISON_TEST on list  $\mathcal{T}$ 
/* Testing the new regions only */
```

RETURN
END SUBDIVIDE(\mathcal{R})

PROCEDURE MULTIPLE_COMPARISON_TEST(\mathcal{X})

/ This tests whether the regions \mathcal{R} on the list \mathcal{X} differ statistically with respect to their means. */*

```

If  $\mathcal{R}^*$  is not in  $\mathcal{X}$  then add  $\mathcal{R}^*$  to  $\mathcal{X}$  and call the new list  $\mathcal{X}'$ 
If homogeneity_of_variance( $\mathcal{X}'$ ) is true then
    Perform Tukey-Kramer multiple comparison test on list  $\mathcal{X}'$ 
else
    Perform Scheffe multiple comparison test on regions on list  $\mathcal{X}'$ 

For each region  $\mathcal{R}$  on  $\mathcal{X}$ :
If  $\mathcal{R}$  is not different from the most preferred region  $\mathcal{R}^*$  then
    If  $\mathcal{R}$  is not on the list  $\mathcal{E}$ 
        Place  $\mathcal{R}$  on the list  $\mathcal{E}$ 
    else
        If  $\mathcal{R}$  is on the list  $\mathcal{E}$  then
            Remove  $\mathcal{R}$  from the list  $\mathcal{E}$ 
        Place  $\mathcal{R}$  on the list  $\mathcal{D}$ 
Next region  $\mathcal{R}$  on  $\mathcal{X}$ :

```

RETURN**END MULTIPLE_COMPARISON_TEST(\mathcal{X})****LOGICAL FUNCTION HOMOGENEITY_OF_VARIANCE(\mathcal{X})**

*/*Define violation as a logical variable that is set to true if normality assumption is violated */*

```

violation = false
For each  $\mathcal{R}$  on  $\mathcal{X}$ :
/* If errors are not normal */
If Shapiro-Wilk test is significant then
    violation = true;
Next  $\mathcal{R}$  on  $\mathcal{X}$ .
homogeneity_of_variance = true.

If violation = false then
    if Bartlett-Box test is significant then
        homogeneity_of_variance = false;
    else
        if Levene's Median test is significant then
            homogeneity_of_variance = false.

```

RETURN**END HOMOGENEITY_OF_VARIANCE(\mathcal{X})**

PROCEDURE SAFETY_NET(\mathcal{D})

```
Call the smallest  $\Delta$  encountered for any region so far  $\Delta_{\min}$ .
Take region  $\mathcal{R}$  at the front of  $\mathcal{D}$ .
/* Call the  $\Delta$  for this region  $\Delta_{\mathcal{R}}$ . */

If stop_criteria ( $\mathcal{R}$ ) then
    RETURN.

Get the best region found so far,  $\mathcal{R}^*$ .
If  $\Delta_{\mathcal{R}} \geq 4\Delta_{\min}$  then
    If sig_better ( $\mathcal{R}, \mathcal{R}^*$ ) then
        Place region  $\mathcal{R}$  on the list  $\mathcal{E}$ 
    else
        Perform BUMPINESS on the region  $\mathcal{R}$ ;
else
    If sig_better ( $\mathcal{R}, \mathcal{R}^*$ ) then
        Place region  $\mathcal{R}$  on the list  $\mathcal{E}$ 
    else
        REMOVE region  $\mathcal{R}$  from the list  $\mathcal{D}$ 
```

RETURN

END SAFETY_NET(\mathcal{D})

LOGICAL FUNCTION SIG_BETTER($\mathcal{R}, \mathcal{R}^*$)

```
/* If  $\mathcal{R}$  is significantly better than  $\mathcal{R}^*$ , then sig_better is true. */
Let  $\mathcal{X}$  = list comprised (only) of  $\mathcal{R}$  and  $\mathcal{R}^*$ .

If homogeneity_of_variance ( $\mathcal{X}$ ) then
    Perform TUKEY-KRAMER comparison test on  $\mathcal{X}$ 
else
    Perform SCHEFFE comparison test on  $\mathcal{X}$ .

If appropriate test above ( $\mathcal{R} > \mathcal{R}^*$ ) is significant then
    sig_better = true
else
    sig_better = false.
```

RETURN

END SIG_BETTER($\mathcal{R}, \mathcal{R}^*$)

PROCEDURE BUMPINESS(\mathcal{R})

/* To get to this point, \mathcal{R} has been shown to be inferior to \mathcal{R}^* . But \mathcal{R} still covers a relatively large area, i.e., it still has a large Δ relative to Δ_{\min} . */

REMOVE region \mathcal{R} from the list \mathcal{D} .

Bisect the region \mathcal{R} along each axis.

Make sure all runs for all gridpoints have been recorded.

For each gridpoint that does not have runs, run replications (e.g., three).

Calculate bumpiness (an approximation to the second derivative) in the direction of all axes as follows:

$$f''(a) \approx \frac{f(a-h) - 2f(a) + f(a+h)}{h^2}$$

If \mathcal{R} has significant bumpiness in the direction of optimization then

/* if the problem is maximization and the bumpiness indicates a maximum or */

/* if the problem is minimization and the bumpiness indicates a minimum, then */

For each subregion \mathcal{R}_i of \mathcal{R}

if not **stop_criteria**(\mathcal{R}_i) then

PLACE subregion \mathcal{R}_i on the list \mathcal{E} .

next \mathcal{R}_i .

RETURN

END BUMPINESS(\mathcal{R})

VITA

The author was born in Manila, the Philippines, on February 25, 1962. He graduated from the Ateneo de Manila Grade School (Honors Program) in 1976, Ateneo de Manila High School (Honors Program) in 1980 and Ateneo de Manila University with a Bachelor of Science in Management Engineering (Honors Program) in 1984. He finished his MBA (concentration in Management Science) at Virginia Polytechnic Institute and State University in 1988.

He worked in the Computer Science Department and Management Science Department as a graduate teaching assistant. He was accepted into the doctoral program in Management Science prior to completion of his MBA. As a doctoral student, he continued as a graduate teaching assistant and also was a part-time lecturer in the Management Science department. He graduated with a Doctor of Philosophy degree in Management Science on November 1, 1995.

The author is currently working as an Operations Research Analyst at RV Industries in Atlanta, GA.

Fernando C. Giochi