A Knowledge-Based Simulation Optimization System with Machine Learning

by

Ingrid W. M. Crouch

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Management Science

APPROVED:

Loren P. Rees

Loren P. Rees, Chairman

[illegible]

Allen G. Greenwood

Terry R. Rakes

Terry R. Rakes

[illegible]

Robert T. Sumichrast

[illegible]

Bernard W. Taylor, III

May 6, 1992

Blacksburg, Virginia

A Knowledge-Based Simulation Optimization System

with Machine Learning

by

Ingrid W. M. Crouch

Loren P. Rees, Chairman

Management Science

(ABSTRACT)

A knowledge-based system is formulated to guide the search strategy selection process in simulation optimization. This system includes a framework for machine learning which enhances the knowledge base and thereby improves the ability of the system to guide optimizations. Response surfaces (i.e., the response of a simulation model to all possible input combinations) are first classified based on estimates of various surface characteristics. Then heuristics are applied to choose the most appropriate search strategy. As the search is carried out and more information about the surface becomes available, the knowledge-based system reclassifies the response surface and, if appropriate, selects a different search strategy. Periodically the system's Learner is invoked to upgrade the knowledge base. Specifically, judgments are made to improve the heuristic knowledge (rules) in the knowledge base (i.e., rules are added, modified, or combined). The Learner makes these judgments using information from two sources. The first source is past experience -- all the information generated during previous simulation optimizations. The second source is results of experiments that the Learner performs to test hypotheses regarding rules in the knowledge base.

The great benefits of simulation optimization (coupled with the high cost) have highlighted the need for efficient algorithms to guide the selection of search strategies. Earlier work in simulation optimization has led to the development of different search strategies for finding optimal-response-producing input levels. These strategies include response surface methodology, simulated annealing, random search, genetic algorithms, and single-factor search. Depending on the characteristics of the response surface (e.g., presence or absence of local optima, number of inputs, vari-

ance), some strategies can be more efficient and effective than others at finding an optimal solution. If the response surface were perfectly characterized, the most appropriate search strategy could, ideally, be immediately selected. However, characterization of the surface itself requires simulation runs. The knowledge-based system formulated here provides an effective approach to guiding search strategy selection in simulation optimization.

Acknowledgements

First, I would like to thank my chairman and advisor Professor Loren Paul Rees; I am indebted to him for his work with me during my graduate studies. He has always been encouraging, helpful, willing to share his ideas with me, and fun to work with. He has taught me much -- both directly and by example -- about research, teaching, and service to others, which I will take with me into my professional and personal life.

I would like to thank Dr. Allen G. Greenwood for sharing his ideas with me, for helping to refine the content and presentation of this dissertation, and for serving on my committee.

I would like to thank Professor Bernard W. Taylor, III, Head of the Department of Management Science, for providing facilities and financial support during my graduate studies, and also for serving on my committee.

I would like to thank Dr. Terry R. Rakes and Dr. Robert T. Sumichrast for their contributions to my education and for serving on my committee.

I would like to thank Dr. Lance A. Matheson for substituting for Dr. Greenwood during my oral comprehensive exam and my final defense.

I would like to thank Gerry Chenault, Tracy McCoy, Sylvia Seavey, and Teena Long for help and assistance in numerous ways during my time in this department.

Renato Panis, Lars Wiegmann, Barry Wray, Jack Leu, Fernando Siochi, Mark Coffin, Wendy Ceccucci, Gerald Kohers, Chris McCart, and especially Ina Samanta Markham have greatly enhanced my years in the Management Science department.

Special thanks go to my parents, Tor and Bergljot Moksvold, and my sister, Karen Moksvold, for their encouragement and support throughout my education.

Finally, I would like to thank my husband, Jeffrey Crouch, for constant encouragement, support, help, and fun during these years of graduate study.

Table of Contents

List of Illustrations

List of Tables

Chapter 1: Introduction

Simulation plays a critical role in many fields, from traffic modeling to personnel and job shop scheduling to military campaign planning. Simulation permits 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 run through a simulation model, the simulation model output(s) provides an estimate of how the true system would respond to these inputs. A response surface is defined as the response of the simulation model to all possible input combinations. For many applications it is desired to find the best possible (i.e. optimum) response or attain specific goals for the response of the system. Although simulation is very useful in predicting the system outcome or response for a given set of input conditions, it does not of itself indicate the input conditions required to achieve the desired outcome. The process of finding the input conditions that will yield the optimal (or near-optimal) system response is referred to as simulation optimization, and can be very expensive and time consuming.

Earlier work in simulation optimization has led to the development of different search strategies for finding optimal-response-producing input levels. These strategies include, among others, response surface methodology, simulated annealing, random search, genetic algorithms, and single-factor search. Depending on the characteristics of the response surface (e.g., presence or absence of local optima, number of inputs, variance, etc.), some strategies can be more efficient and effective than others at finding an optimal solution. The great benefits of simulation optimization (coupled with the high cost) have highlighted the need for efficient algorithms to guide the selection of search strategies. If the response surface were perfectly characterized, the most appropriate search strategy could, ideally, be immediately selected. However, characterization of the surface itself requires simulation runs.

In this work a knowledge-based system is formulated to guide the strategy-selection process in simulation optimization. A scheme for classifying a response surface and then applying heuristics to choose the most appropriate search strategy is described. As the search progresses and more information about the surface becomes available, the knowledge-based simulation optimization system (KBSOS) reclassifies the response surface and changes the search strategy accordingly.

A framework for machine learning in the context of this knowledge-based simulation optimization system is also presented. The goal of the "learner" is to improve the ability of the KBSOS to guide future optimizations. Specifically, the heuristic knowledge (rules) in the knowledge base is improved (e.g., rules are added, modified, or combined). The learner makes judgments using information from two sources. The first source is past experience -- all the information generated during previous simulation optimizations. The second source is results of experiments that the learner performs to test hypotheses regarding KBSOS rules.

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

Simulation and 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 1.1 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.

The usefulness of 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 field of choices. Safizadeh (1990) discusses a variety of strategies and their application. He concludes that generally RSM approaches are the 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

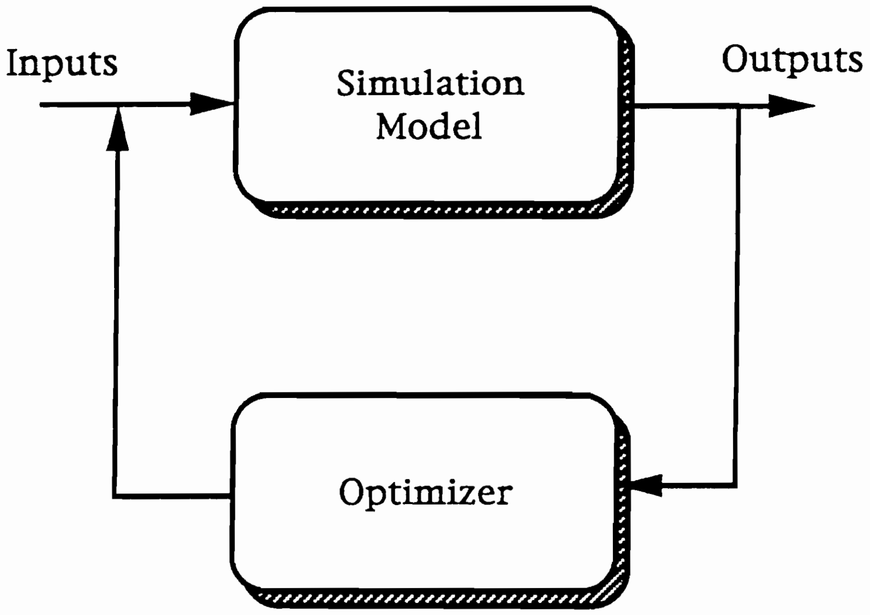


Figure 1.1: The simulation-optimization process

surfaces. He found that the relative effectiveness of the 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 two conclusions. First, organized guidance is needed to help users choose appropriate search strategies. Safizadeh 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." Second, research findings may be translatable into heuristics which could be used in an expert or knowledge-based system to guide strategy selection.

Not only are heuristics needed for selecting search strategies, they are also used in carrying out the strategies. Consider, for example, the search strategy RSM. Many judgments are necessary to implement this strategy; among them are: choice of experimental design, region of fit, step size, and when to change search direction. It is possible that these types of heuristics could also be included in a knowledge-based (expert) system.

Knowledge-Based (Expert) 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 1.2. 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.

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 proposed knowledge-based system will provide guidance for carrying out simulation optimizations, it will not 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 its software over time (learns). How this

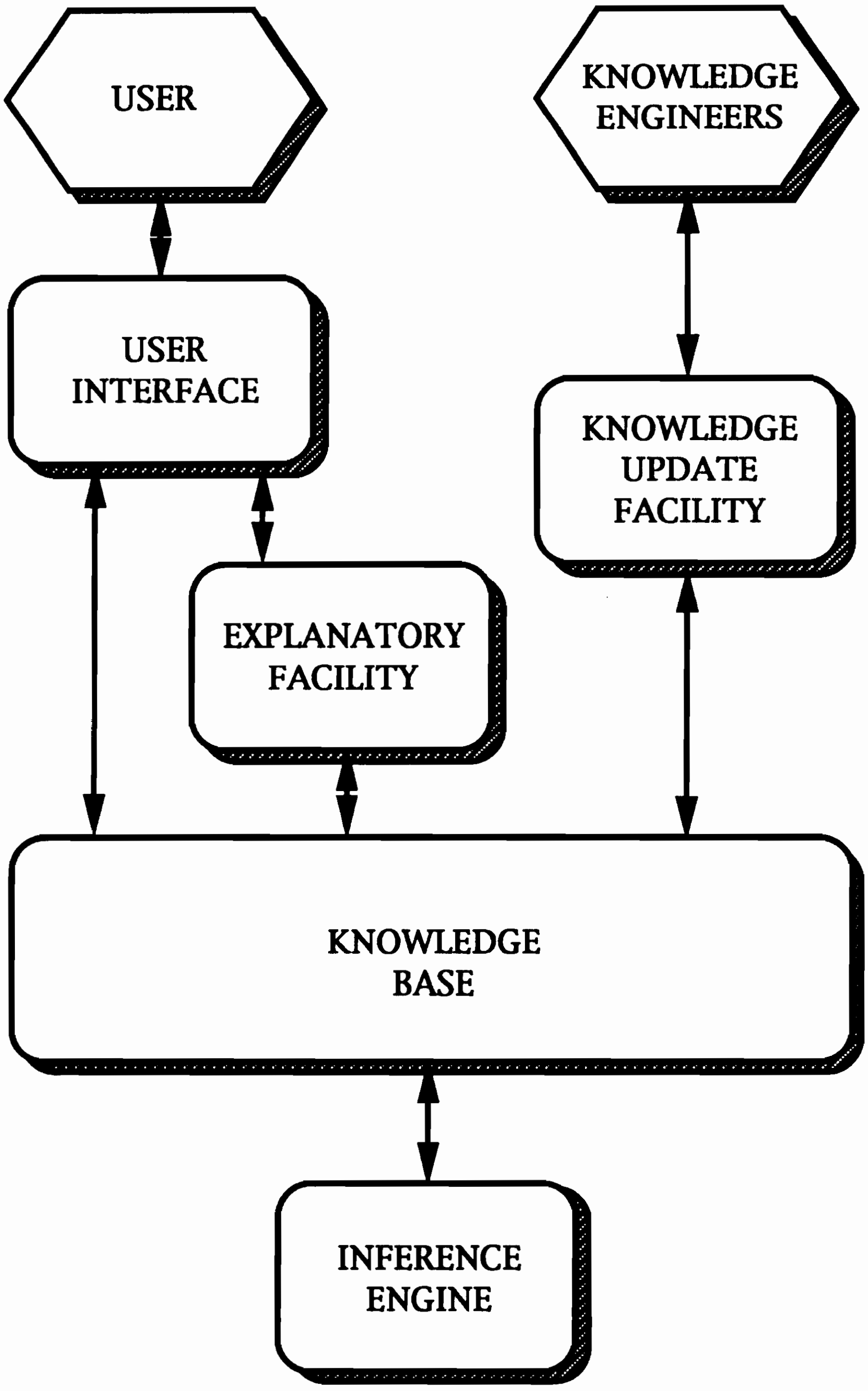


Figure 1.2: Expert system architecture

can be done for simulation optimization will be discussed in a later chapter; 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.

Framework

Greenwood, Rees, and Crouch (GRC) (to appear) present an architecture for a knowledge-based system which separates the heuristic knowledge imbedded in simulation optimization from fixed procedural aspects. As shown in figure 1.3, this architecture has three components.

The key component is the Knowledge Kernel, since knowledge -- heuristic and procedural -- is the essence of the system. The Knowledge Kernel contains facts (data) about different simulation optimization cases, analytical procedures, and heuristic rules to guide the simulation-optimization process.

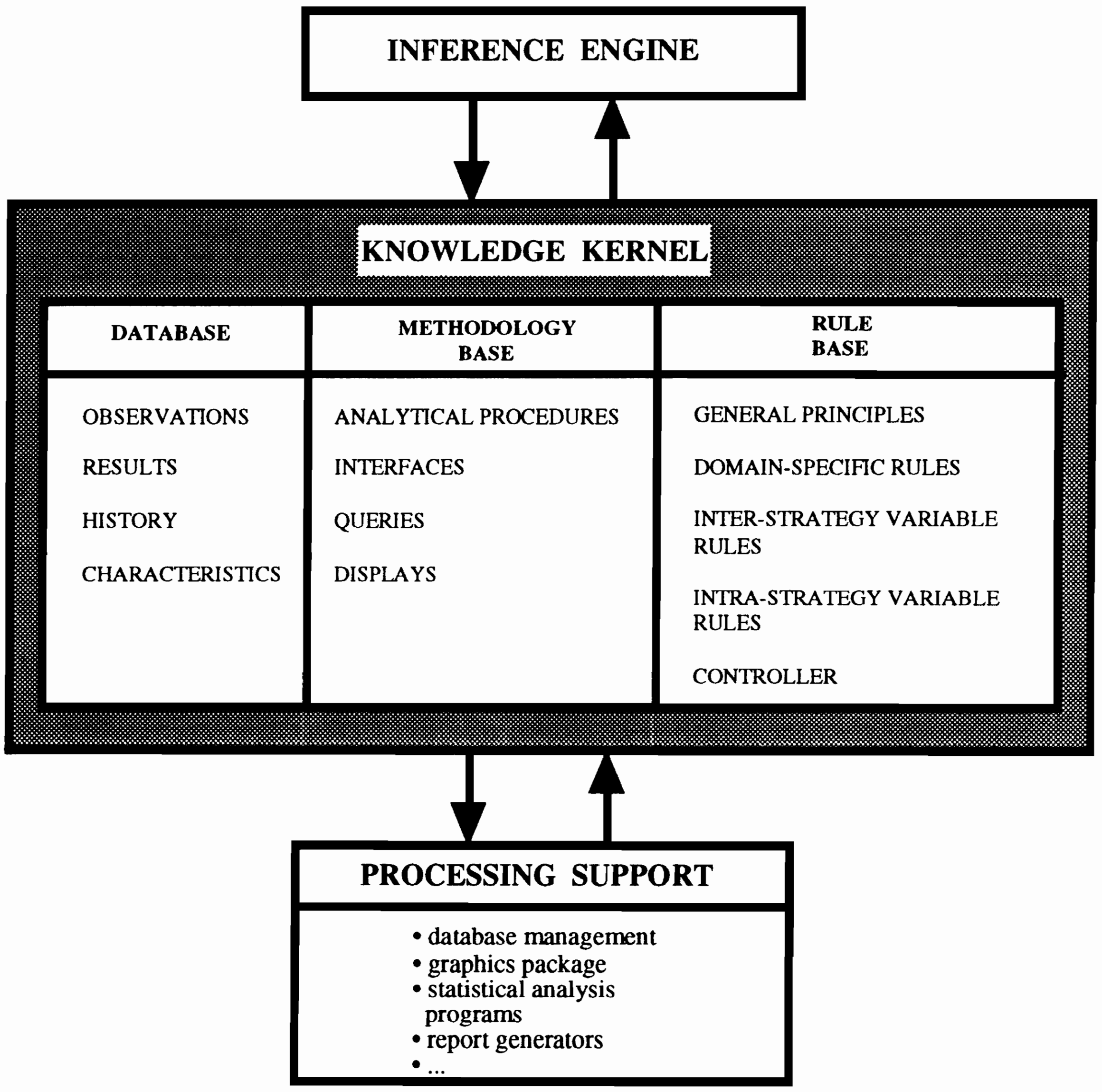


Figure 1.3: Greenwood-Rees-Crouch simulation optimization architecture

The Inference Engine draws conclusions and selectively "fires" rules stored in the knowledge base.

Processing Support provides software for such needs as database management, graphics, and reports.

GRC also present a framework for machine learning in the simulation optimization context, in which the learner would use information from the database portion of the knowledge kernel to make inferences about the art of simulation optimization and then modify the rule base accordingly. They discuss different types of learning which could be used to improve the heuristics in the knowledge kernel.

However, GRC do not explicitly stipulate how to decide which search strategies should be used under which simulation conditions, or how to organize and perform the different learning types in this context. These issues are the subject of this dissertation.

Purpose of Research

Although Greenwood, Rees and Crouch allowed for strategy selection in their knowledge kernel, they gave no prescription for how to determine which search strategy should be applied to a given surface; they merely specified that rules be used to select a strategy. Also, there was no overall organized scheme elucidated for determining the mapping between response surfaces and search strategies. It is these major gaps or limitations that are now addressed.

This research suggests a classifier approach to specifying the response-surface to search-strategy mapping. The first contribution is a methodology whereby simulation response surfaces are classified using a set of characteristics specifically chosen to differentiate between the available search strategies. In short, a classifier categorizes a given surface, and then heuristics (stored in a knowledge base) recommend a specific search strategy. This is demonstrated with an example response surface.

The second contribution of this research is the design of a learner tailored for the knowledge-based simulation optimization system discussed above. Greenwood, Rees and Crouch introduced the concept of machine learning in the simulation optimization domain. In this work their ideas are developed further -- the architecture of the learner is revamped and a process by which learning can take place is presented.

Scope and Limitations

The knowledge-based simulation optimization system described will provide guidance for optimization of simulation models with continuous inputs. (Capability to deal with discrete inputs can be added in the future.) However, it does not consider every possible classification characteristic or search strategy; which other characteristics and strategies would be useful is an important subject for future study. The learner does, however, provide machanisms for introducing new classification characteristics and search strategies to the KBSOS.

The learner design is discussed in detail, and some aspects are demonstrated. A fully-functioning learner is not demonstrated however; this would require a large library of simulation optimization cases and is beyond the scope of this dissertation.

Plan of Presentation

The next chapter surveys related literature, and especially delves more deeply into the work by Greenwood, Rees and Crouch. Chapter three describes the knowledge-based simulation optimization system, which uses heuristics to select most appropriate search strategies based upon the classification and periodic reclassification of response surfaces. Chapter four presents the learner, which is designed to improve the heuristic knowledge in the simulation optimization system of the previous chapter. Chapter five summarizes contributions and presents a plan for furthering the state of the art in a learning knowledge-based simulation optimization system.

Chapter 2: Literature Review

The Simulation Optimization Problem

The simulation-optimization process is exemplified in figure 2.1 by two "black boxes." The first box represents the simulation model, which combines n controllable input factors (represented by the X_i's) in a manner that approximates the behavior of some real system and produces a set of m output or response values (represented by the Y_j's). The responses are typically measures of merit or performance of the system being modeled and are stochastic or exhibit random variation (mostly as a result of a set of uncontrollable factors not directly considered in the model). The responses can be considered as functions of the controllable inputs and a random error term (represented by the ε_j's); i.e., there are m distinct $(n+1)$-dimensional hypersurfaces, each having its own characteristic random variation.

The second box in figure 2.1 represents the optimizer, which determines the scenario (the value of the input factors or X's) that will result in the "best" response, i.e., find the value of the decision variables that "optimize" in some sense the output of the system. The optimizer drives the search/experimentation process based on a set of user-supplied goals/criteria. For example, a mil-

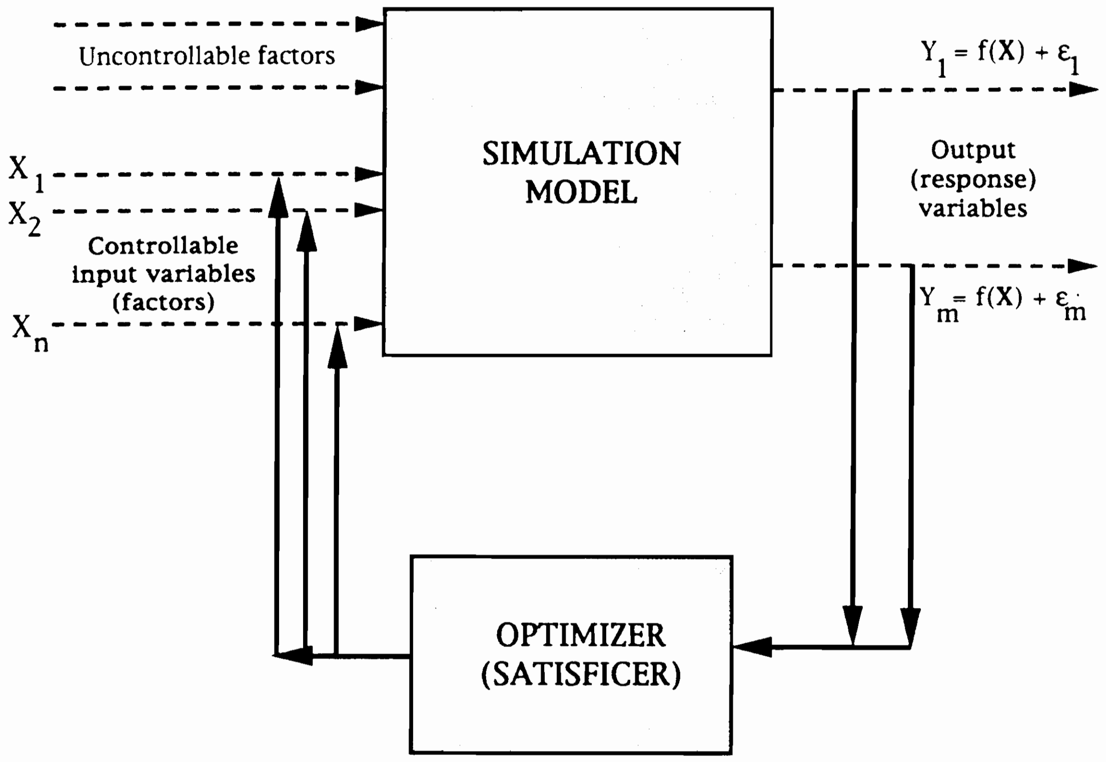

Uncontrollable factors
X_1
X_2
Controllable input variables (factors)
X_n
SIMULATION MODEL
$Y_1 = f(X) + \varepsilon_1$
Output (response) variables
$Y_m = f(X) + \varepsilon_m$
OPTIMIZER (SATISFICER)

Figure 2.1: The Simulation Optimization Process

itary analyst searches for the best combination of planes and tanks (controllable input variables X_1 and X_2) that result in minimum casualties, maximum territory captured and greatest materiel damage inflicted on the enemy (output responses Y_1, Y_2, and Y_3).

Typically, problems represented in a simulation framework cannot be stated in an explicit mathematical expression; this precludes the application of standard optimization methods. Since the true response surfaces are usually unknown, they are most often approximated by a polynomial function fitted to the output data generated by the simulation. In fact, true optimization is usually unobtainable; a more pragmatic approach is to locate an improved solution or satisfactory level of goal attainment, called satisficing. Therefore, the optimizer depicted in figure 2.1 is often referred to as a "satisficer." Hereafter, our reference to a satisficer will mean any automated optimizer or satisficer.

One set of statistical tools that has been extensively applied to the problem of simulation optimization is response surface methodology (RSM). It is a sequential approach that is both an adaptive (observed response influences the value of subsequent controllable factors) and derivative-based (establishes the direction of search) approach. Its beginnings are traced to the work of Box and Wilson (1951) on determining optimum operating conditions in the chemical industry. Myers, Khuri, and Carter (1989) provide a more general definition of RSM: "... a collection of tools in design and data analysis that enhance the exploration of a region of design variables in one or more responses." For more information on RSM, the interested reader is referred to: Box and Draper (1987), Brightman (1978), Khuri and Cornell (1987), Mead and Pike (1975), Myers (1971), and Myers, Khuri and Carter (1989).

In order to lay the groundwork for a discussion of the problems inherent in the simulation-optimization process, the following section provides a brief overview of how RSM is used to optimize simulation results. RSM is not, of course, the only search technique used for simulation optimization; however, for the sake of brevity, most examples and references in this chapter will use RSM to illustrate simulation optimization issues and concepts.

The Simulation Optimization Process Using RSM

The application of RSM to the problem of simulation optimization is outlined in this section. It is illustrated through the software program SATSIM (SATisficing SIMulation) by Rees, Clayton and Taylor (1985). We are not touting SATSIM as the best way to solve the simulation satisficing or optimization problem. Rather it is used as a means to illustrate the point that this process is a coupling of "art" and "science." SATSIM is only one implementation of RSM applied to simulation; it happens to be the one most familiar to us.

The general idea in SATSIM is to place a feedback loop around a simulation model and adjust the input factors of the simulation model based on a growing history of simulation responses or outputs (as was depicted in figure 2.1). The control mechanism in the feedback loop is called a satisficer in SATSIM because the output variables are not all optimized. Instead, each response is compared to a user-specified aspiration level, and the system attempts, if it is feasible, to drive the response level beyond this goal. While most problems in simulation optimization involve multiple responses, there is no standard way to combine the multiple responses. Several approaches are proposed in the literature: Montgomery and Bettencourt (1977), Biles and Swain (1977 and 1979), Rees, Clayton and Taylor (1985). For brevity and with no loss in generality, we assume in all subsequent discussion that only one response/goal is being considered.

The optimization process begins with specifying the relevant variables, parameter values, and initial conditions. Each response (system operating characteristic that is to be optimized or satisfied) and its desired level of achievement is stipulated. The controllable factors (decision variables, inputs, X_i's) are identified along with any limits or operating constraints that need to be considered. Preliminary experimentation or screening may be necessary to reduce the number of factors under consideration. An initial experimental region and a starting point (in terms of the X_i's) are established and SATSIM specifies an experimental design about that point. Since the most common search procedure, steepest ascent, is sensitive to the relative spacing of the levels of the decision variables, they must be appropriately scaled or coded.

Before any searching can begin, observations (samples) must be taken at each design point; i.e., SATSIM executes the simulation model and determines a response value for each input point. Normally it is assumed that these samples are a long way from the optimum and the higher-order polynomial terms of the model used to fit these data are not significant; therefore, a first-order model is sufficient. A plane (or hyperplane) is then fit through the output points using ordinary least squares regression and checked using analysis of variance for lack of fit.

If the least-squares fit is "good" (slopes are significant and the model does not demonstrate a significant lack of fit) a search -- typically steepest ascent when the goal is to maximize the response value -- is begun. SATSIM steps out in the direction of steepest slope as indicated by the regression coefficients and requests that an additional simulation run be made. SATSIM continues to step out in the direction of steepest ascent as long as the simulation output increases. When a decrease is observed, SATSIM returns to its previous point and asks that another experimental design be specified, and the entire sequence repeats.

The above process continues until the top of the surface (regarded as a hill to be climbed in the two-factor case) is achieved, or at least the aspiration level is reached, if either is possible. A series of first-order designs and steepest ascent are used until there is a significant lack of fit, i.e., second-order terms begin to dominate. At this point a second-order design is constructed, usually through augmentation of the first-order design with additional design points. SATSIM executes the simulation model and determines a response value for each design point. A second-order model is then fit through the observations using ordinary least-squares regression and checked using analysis of variance for lack of fit. A canonical analysis is used to examine the nature of the curvature of the fitted surface (max, min, saddle, ridge). Ridge analysis (analogous to the steepest ascent procedure used for first-order designs) can be used with the second-order model to guide the search toward an improved solution.

Problems Inherent in the Process

Although RSM is probably the most widely-used approach for finding the optimal solution to a simulation problem, it has several limiting assumptions, e.g., continuous variables, single unimodal response, polynomial models with normally-distributed error term estimated using ordinary least-squares regression methods. Since it is not a panacea, there are many other approaches reported in the literature. Whereas RSM is considered an adaptive, derivative-based procedure, there are also adaptive non-derivative based search techniques (e.g., coordinate, pattern), non-adaptive methods (e.g., random, grid), and many hybrid approaches. The important point to recognize is that any simulation optimizer should not be limited to any one approach; it should contain many different procedures and a mechanism to provide advice on the appropriate time to apply each procedure. Normally, in order for this to be effective, there needs to be a monitoring device to track the behavior of the observed response and characterize it in such a manner that there can be advice on what to do next.

The simulation optimization process is usually not as straightforward as the one described in the previous section. Typically the decisions at each step are more ambiguous and uncertain and the overall process is more iterative and less sequential; e.g., it is often necessary to return to a first-order design and explore another region, even after the analysis has progressed to the second-order design phase. This is especially true if the response function is complex and/or involves considerable variation.

Within RSM, there are many issues to which there are no clear answers -- issues that rely heavily on judgment and heuristics. Some possible solutions to these problems are reported in the literature, but most reside in the minds of practicing RSM professionals.

One such issue in RSM is which procedure to use at each point in the analysis process, e.g., when should a certain design be constructed, and when should a particular search technique be employed?

While there is not a lot of research published in this area, there has been enough to indicate that some procedures perform better than others under certain conditions. Smith (1973) empirically evaluated search techniques in a decision-theory framework and developed guidelines for their application. In essence, he attempted to find a partial answer to one main question: for a specified number of controllable factors and available computer runs, what is the best search technique to use and how much gain is expected from using this technique on response surfaces having specific characteristics. He found, for example, that single-factor search strategies are nearly always dominated by the performance of other techniques; random search is not necessarily better even when a large number of factors is considered. Along the same line, Montgomery and Evans (1975) explored how well six second-order RSM designs performed on six known surfaces, each considered at six levels of experimental error. The results from both of these studies, as will be discussed later in this chapter, can easily be translated into a rule base for use in an expert system.

Another difficult issue in RSM analysis involves determining the appropriate step size to take in a specified search direction. Normally the step size is chosen in proportion to the spacing between the upper and lower levels of the design. Biles (1975) provides an alternative approach that uses regression. A related issue involves the question of when to stop searching in a specified direction and invest in a new experiment. Most often the search continues until there is a drop in the response; Myers and Khuri (1979) offer an alternative that accounts for the possible occurrence of a false drop in the response.

Yet another problem is how to allocate a limited number of simulation experiments. Daughety and Turnquist (1978 and 1981) present a novel "budget focused" multistage search procedure that allocates simulation experiments both temporally (number of stages and number of experiments in each stage) and spatially (within each stage, the optimal allocation of experiments to some region of search). Another unique feature of their approach is that after each stage's experiments are run, a response surface is estimated using all of the experimental observations made so far. The surface is based on actual simulation results and "pseudo-experiments" generated by cubic spline functions. The resulting surface provides the starting point and search direction for the next stage.

The problems discussed above are not peculiar to RSM. They are inherent generally in any of the specific techniques used to optimize simulations; the use of any single methodology brings with it a set of limiting assumptions. Any single procedure will be more iterative, and less sequential as response functions become more complex. There is a need for heuristics and expertise to be explicitly incorporated into the optimizer -- be it to assist in the technical details of search issues or the specification of design matters.

The above annotation of the problems associated with simulation optimization should make it clear that there are judgments that either the user or a program must make. Such judgmental decisions include: how large to make the region of fit over which the experimental design is placed and the regression performed, what step size should be taken along the direction of steepest ascent, what particular experimental design should be used, etc. These decisions depend, in part, on the cost to the user of each simulation run, which statistical criterion (bias, mean-squared error, variance) is most appropriate, prior knowledge about the response function, practical knowledge on which search technique (Box's complex-search, simulated annealing) works best with which design and in which environment (type of surface, amount of variation), etc.

We believe the judgments and decisions described above to be the "art" and "craft" of RSM. Notice that we have termed these decisions art and not magic or luck. Individuals with years of experience either teaching or consulting in the RSM area consistently appear to have better strategies and success in "getting to the top" of surfaces than do those without this experience. This leads us to believe that there is genuine expertise in the art of RSM that can be acquired, refined, and organized into a knowledge base that can assist conventional programs like SATSIM.

The other aspect of RSM and simulation optimization that should be clear at this point, especially from the overview of the process provided in the previous section, is that any simulation satisficer such as SATSIM utilizes many statistically-based procedures and methodologies, e.g., experimental design, ordinary least-squares regression, analysis of variance, direction of steepest ascent, canonical analysis, and ridge analysis. This is the "science" part of RSM. These are well-defined procedures and algorithms. Conventional (procedural) programming languages lend themselves well to implementing such procedures. In fact, as discussed in the next section, there are many third-generation language subroutines, modules, programs, and packages that exist for these procedures.

The following section provides a brief overview of the current state of simulation optimization software. The review clearly demonstrates that no software explicitly recognizes the heuristic and algorithmic aspects of the problem. It also is clear that no prior framework or architecture has addressed this concern.

Simulation Optimization Related Software

Myers et al. (1989) note there is a severe shortage of general RSM software. This is also true, until recently, of the general area of statistics that is related to the design of experiments, at least in comparison to data analysis software. Nachtsheim (1987), in his review of computer-aided design tools, notes that interactive experimental design software has just recently started to emerge; some packages he describes include tools for optimizing response surfaces.

The idea of an integrated simulation satisficer is certainly not new -- there are several reports of such systems in the literature. Smith nearly 15 years ago outlined, in several articles (1973, 1974, and 1976), the need for an automated optimizer for computer simulations. He (Smith, 1973) identified the need to include efficient strategies (search technique, screening method, variance reduction) and replace the existing trial-and-error procedures. The satisficer program, described in Smith (1975), included not only design and regression algorithms but heuristics that governed their use. Unfortunately the links between the algorithms and the rules for their application were "hardwired"; of course, based on the computer technology of the time, this was to be expected.

All of the other simulation optimization software reported in the literature is similar to that described by Smith, at least in terms of their framework. They are written in third-generation languages, with algorithms and heuristics bundled within the computer code. For example, Pegden and Gately (1980) incorporated a FORTRAN-based optimization routine (a non-derivative based pattern search) into the simulation language SLAM. Bengu and Haddock (1986) combined a simulation generator (translates a description of a system into a SIMAN-based simulation program) and an optimization subroutine. Their generator is limited to continuous-review inventory systems; and their satisficer is limited to finding the optimal inventory-control policy using single and multivariable, derivative-free, unconstrained search procedures. Rees, Clayton, and Taylor (1985), in an interactive computer program called SATSIM, combined the techniques of RSM with goal programming concepts to obtain satisfactory solutions to multiple-response simulation models.

All of the aforementioned programs have several limiting features in common; they: utilize a few solution approaches, incorporate few heuristics (the heuristics that are included are built in), are not easily adapted to conditions that are not pre-planned, are not easily modified, and intermingle control and data. These characteristics should not reflect negatively on the systems' developers; they used the computer technology that existed at the time.

The main problems with SATSIM and other available programs pertain to the storage of heuristic knowledge. The first problem is that it is very cumbersome to store heuristic knowledge in a third-generation language. The heuristics used to provide guidance to SATSIM are quite complex; fifth-generation languages and expert system shells now exist that can represent knowledge much more naturally. The second problem is related to the first: the art and science of simulation optimization are so intermingled that it is not wise or practical to insert the needed "artsy" heuristics into the middle of the "scientific" code. Any attempt to do so, particularly in a learning environment (where new heuristics would be added automatically as experience is gained), would result in unstructured, non-modularized, and probably undecipherable computer code.

What is needed is an explicit recognition of the dual nature of simulation optimization -- the algorithmic and heuristic aspects of the problem. Once this is realized, it is clear current programs must be "unbundled." Artificial Intelligence technology can facilitate this, as is discussed in the following section.

Applying AI Technology to Simulation and Statistical Analysis

Artificial Intelligence (AI) technology in general, and expert systems (ES) in particular, are starting to be applied to a variety of problems in simulation and statistical analysis. Since simulation optimization is the focus of this work and since that process depends heavily on the use of statistical tools, this section briefly reviews the application of AI technology to both of these related areas.

Simulation and expert systems are being combined for a variety of reasons. An expert system can be used as a front-end processor for model formulation or model generation, as an aid in debugging, and as a back-end processor for statistical interpretation and monitoring. In this context, the simulation optimizer described in this chapter would be considered a back-end processor.

Shannon, Mayer, and Adelsberger (1985) discuss the evolving new field of AI-based expert simulation systems. They note that simulation models and languages have contained many ideas presently being used in AI; e.g., in both cases, entities carry descriptive attributes that can be changed dynamically (analogous to frames), entity flow through the system can be dynamically modified (analogous to production rules), and knowledge is represented as a network. There are also several distinct differences. For example, simulation involves numeric processing functions, integrated control and information, explicit solution steps, and programs that operate on user's instructions.

In contrast, AI involves symbolic processing functions, separated control and information, solution steps that involve pattern-invoked searches, and programs that utilize built-in expertise and the ability to learn and modify themselves as needed.

O'Keefe (1986) presents a taxonomy of four ways to combine simulation and ES. The first alternative is to imbed either the simulation tool or the ES tool within the other. A second approach is to separate the software so that they operate in parallel, yet interact with each other. In both of these approaches, the user interacts with only one of the tools. A third possibility is for both tools to cooperate in order to accomplish some task (they may share data). The user may directly interact with both software or interact through a larger enveloping software environment. The final way to combine the two tools is to use the ES as an intelligent front end, i.e., place it between the simulation package and the user. In this context, the simulation optimizer that we propose fits best into the third scheme, where the two modeling tools interact in a cooperative environment.

Finally, Reddy, Fox, and Husain (1986) described a KBS (Knowledge-Based Simulation) system that focuses on automating the simulation life-cycle, and in particular automating the analysis of results -- it can conduct experiments and rate scenarios to make a recommendation. The KBS system is an interpreter that accesses a model and provides simulation, model checking, and data analysis.

Since a simulation optimizer must include a wide variety of statistical procedures, it is important to survey the research directed at applying AI technology and ES to the general area of statistical analysis.

AI technology is starting to be applied to general statistical analysis software. Remus and Kottemann (1986) discuss the movement toward developing an intelligent DSS for statistical analysis. They indicate that the current state-of-the-art statistical packages (e.g., SPSS, SAS, MINITAB) require the user to provide the data, choose the appropriate statistical technique and give the analysis commands, and interprets the results. They indicate that many of the judgments involved in this process -- e.g., what tests to use, how to properly conduct the tests -- can be formalized using artificial intelligence techniques.

Gale and Pregibon (1985) identify two tasks in which statisticians are being assisted by expert system techniques -- data analysis (application of statistical tools to a particular set of data to reach some conclusion) and experimental design (planning the collection of data so that it can be easily analyzed to reach a conclusion). They identify the area of intelligent interfaces (provide guidance, interpretation, and instruction) to the many existing statistical software packages as one potentially fruitful area for AI to contribute to data analysis. They also indicate that AI can be used to help structure some areas of statistical knowledge that are not yet well formalized -- e.g., how methods are chosen and applied to analyze data in practice (a process they called strategy).

Hahn (1985) proposes three general levels of intelligent statistical software: (1) computerized answering and referral service (computerized encyclopedia or indexes to direct users to sources of information), (2) expert guidance embedded in statistical programs (e.g., determining the appropriate model or transformation, explaining the meaning or significance of certain terms, directing the user's attention to special features of the data, suggesting further evaluations or alternative analyses), and (3) automated statistical consulting and data analysis (guide the analyst to a solution of a general statistical problem). He also describes some of the issues involved with their development and some potential future directions. In terms of Hahn's taxonomy, the simulation optimizer that we describe best fits into the second category.

One statistical analysis software package, Data Desk Professional developed by Velleman and Velleman (1988), has incorporated some knowledge on how statistical methods work together.

Greenwood, Rees, and Crouch (to appear) present a new simulation optimization framework which incorporates AI technology. This is described further in the next section.

Unbundling the Process

The strategic overview of the simulation-optimization process presented in Greenwood, Rees and Crouch (GRC) (to appear) provides the means to handle the differing needs of both the heuristic ("art") and procedural ("science") aspects of the problem, yet permits the two to interact in a flexible and responsive environment. This approach is supportive of, and more congruent with, the general analysis process than any alternative proposed to date. As such, it is the foundation for this dissertation, and will therefore be discussed below in some detail.

The GRC framework requires the simulation-optimization process to be viewed in a totally different manner. Unlike the work that has been done to date at the micro level -- e.g., improved designs, search techniques -- this research is aimed at the macro or strategic level. The first part of the framework considers the simulation optimization process from a strategic overview perspective and addresses two basic issues: what are the fundamental variables in the simulation optimization process, and what is the analysis environment in which these variables are explored.

In order to identify the primary variables in simulation optimization, the process is first defined in terms of a single goal: determine the best 'move' to make (place to explore or experiment) given the present location (in terms of level of response and region explored), what has occurred up to this point (previous steps taken and results), and what resources are available. This leads to the identification of four primary heuristic factors that are fundamental to the simulation optimization process: search technique, experimental design, step size, and region of fit; these are referred to as the "strategy variables."

The values of the strategy variables are established in an "analysis environment" that utilizes a variety of techniques and approaches from two broad domains. The first domain, referred to as the science of simulation optimization, is characterized as static, structured, algorithmic, and procedural. The second domain, referred to as the art of simulation optimization, is characterized as dynamic, unstructured, heuristic, and non-procedural. The GRC framework unbundles these two domains. The analysis environment will be explored more fully in the next section, which deals with representing the simulation optimization process in a knowledge-base context.

The relationship between the strategy variables and analysis environment is illustrated in figure 2.2. The top portion represents the analysis environment with its two domains, the art and science of simulation optimization. The two domains are used to establish the value of the strategy variables, as shown in the middle portion of figure 2.2. The strategy variables are then used to define the parameters of a specific simulation experiment(s). The results obtained from executing the simulation model are stored in a database and subsequently used to determine the next operation.

At the most fundamental level, the simulation optimization process is driven by the four strategy variables. The variables constitute two general operations: "design" and "search". Both operations are used to specify where experiments are to be performed, i.e., the specification of the factor levels where the simulation is to be executed in order to measure the response(s). The two operations differ in that the first involves multiple observations to gather information about a region; the second uses one (or sometimes several) observations to test for improvement.

As shown in figure 2.2, the two operations and four strategy variables are highly interrelated and depend on many sources of information from the analysis environment, including process history and prior results (e.g., experimental design, regression analysis, search techniques), and heuristics, expertise, and judgments (e.g., how all of the entities are related, what has been successful in the past). The information from these sources may be conveyed in a variety of displays or views, depending on whether the information is being conveyed to a human user or to logical objects within a computer system.

The first operation, referred to as "design," defines the locations where a set of simulations is to be performed in order to gather information on the immediate environment. The two strategy

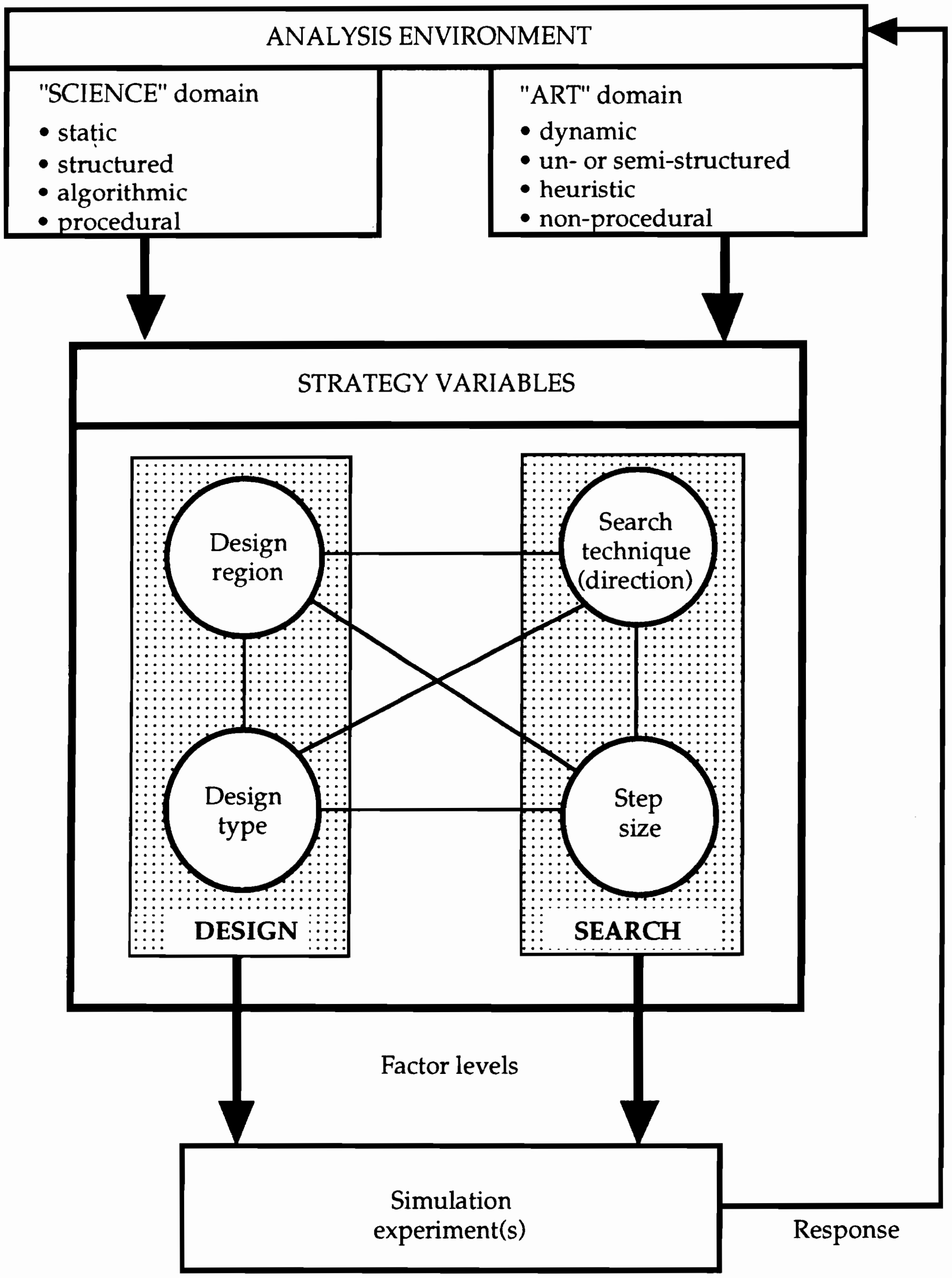


Figure 2.2: Strategic overview of the simulation process

variables associated with the design operation are the size of the design region and the type of design to construct (random, factorial, central composite (CCD), etc.).

The second operation, referred to as "search," uses the sample information from the design operation to explore in a promising new direction. As mentioned above, the search operation normally involves a single simulation experiment (of course a series of searches is possible). The two strategy variables related to search are the type of technique to use in order to determine the search direction (steepest ascent, Box's complex, ridge analysis, etc.) and the step size to take in that direction.

An important benefit of the unbundled satisficer is that this design promotes modularity, as will be seen even more clearly in the remainder of this chapter. New and different procedural routines (i.e., experimental designs and searches) can be "plugged into" the satisficer along with the heuristics that suggest when to invoke them.

To illustrate the GRC framework, consider a simple example. First suppose that, based on the past operations, the satisficer prescribes the next operation to be design. In particular assume the design-type strategy variable is currently set to the value "CCD." Also suppose that based on a previous operation it is inferred that a full design is not necessary and the CCD can be constructed by augmenting an earlier first-order design, thereby defining the design-region strategy variable. These decisions are made by both domains in the analysis environment -- the art domain utilizes expertise and heuristics to decide what to do; the science domain utilizes procedures and algorithms to decide how it should be done.

Once the values of the strategy variables have been set, several procedures are executed. First, a library procedure is executed to specify where the new simulation runs need to be made. (An interface program handles the protocol between the particular experimental design program and the database.) The design information is passed to a simulation program, the experimentation runs are made, and the responses are recorded in the satisficer's database. Note that another interface program is used to link the general satisficer with specific computer simulation models and packages

(GPSS, SLAM, SIMSCRIPT, etc.). That interface contains the protocol for extracting the factor settings for each experiment from the database and translating them to a form readable by the simulation program. It also contains the protocol to receive response values from the simulation and record them in the satisficer's database.

Finally (for the purposes of this example), after the simulation runs have been completed, regression and ANOVA procedures are called to fit a second-order surface and test it for lack of fit. The results are passed along to the analysis environment and are used to determine the next operation and the new values of the strategy variables.

The GRC framework is designed to specifically address the problems described earlier that are inherent in the simulation optimization process. It embraces the application of a wide variety of techniques and approaches. It truly supports the analysis process; i.e., it does not drive it, nor does it force the process to be procedural and linear in nature. It also exploits the application of Artificial Intelligence (AI) technologies; i.e., it calls for intelligence to be placed around or among statistical procedures and not into existing statistical programs. It provides a fresh look at how simulation optimization software should be structured and designed and identifies existing computer technologies that are available to meet this need. The framework is expanded in the next section by describing the analysis environment in more detail.

Simulation Optimization in a Knowledge-Based Context

This section presents the second contribution of the GRC paper: further development of their framework, focused on the analysis environment that was introduced above. As shown in figure 2.3, the analysis environment is represented in a knowledge-based context by defining it in terms of three components: an inference engine (the problem processor that carries out the reasoning), a knowledge kernel (contains facts and expertise), and a set of processing support software. In terms of the framework, the main component of the analysis environment is the knowledge kernel. It will be described in detail following a brief discussion of the other two components. (These components are dealt with quickly because they are more widely known concepts.)

The inference engine contains general problem-solving strategies that search through available information and rules until a decision or solution is found. Most commercial expert system shells (e.g., GoldWorks, VP-Expert, etc.) contain an inferencing mechanism. Two common reasoning or search strategies used in expert systems are backward and forward chaining.

Processing support software consists of ancillary computer packages and programs that are called from within an expert system. They are specific implementations of tasks needed by the satisficer, e.g., relational database management systems, database query languages, graphics packages, statistical analysis software, report generators, and special procedures or subroutines (written in FORTRAN, C, PASCAL, etc.).

The knowledge kernel component of the analysis environment contains all of the information needed to set the strategy variables and decide what operations are to be performed at what time. Three types of knowledge are represented in the kernel -- facts, expertise, and methodologies; they are stored in a database, a rule base, and a methodology base, respectively. Each of these is discussed in the following subsections. Note that the knowledge representations are discussed in conceptual terms, which may differ from physical representations. (For example, if the knowledge is physically represented in terms of frames, then the database and rule base elements can be combined in the frame context.) For simplicity in this paper, we treat the database and rule base as separate representations.

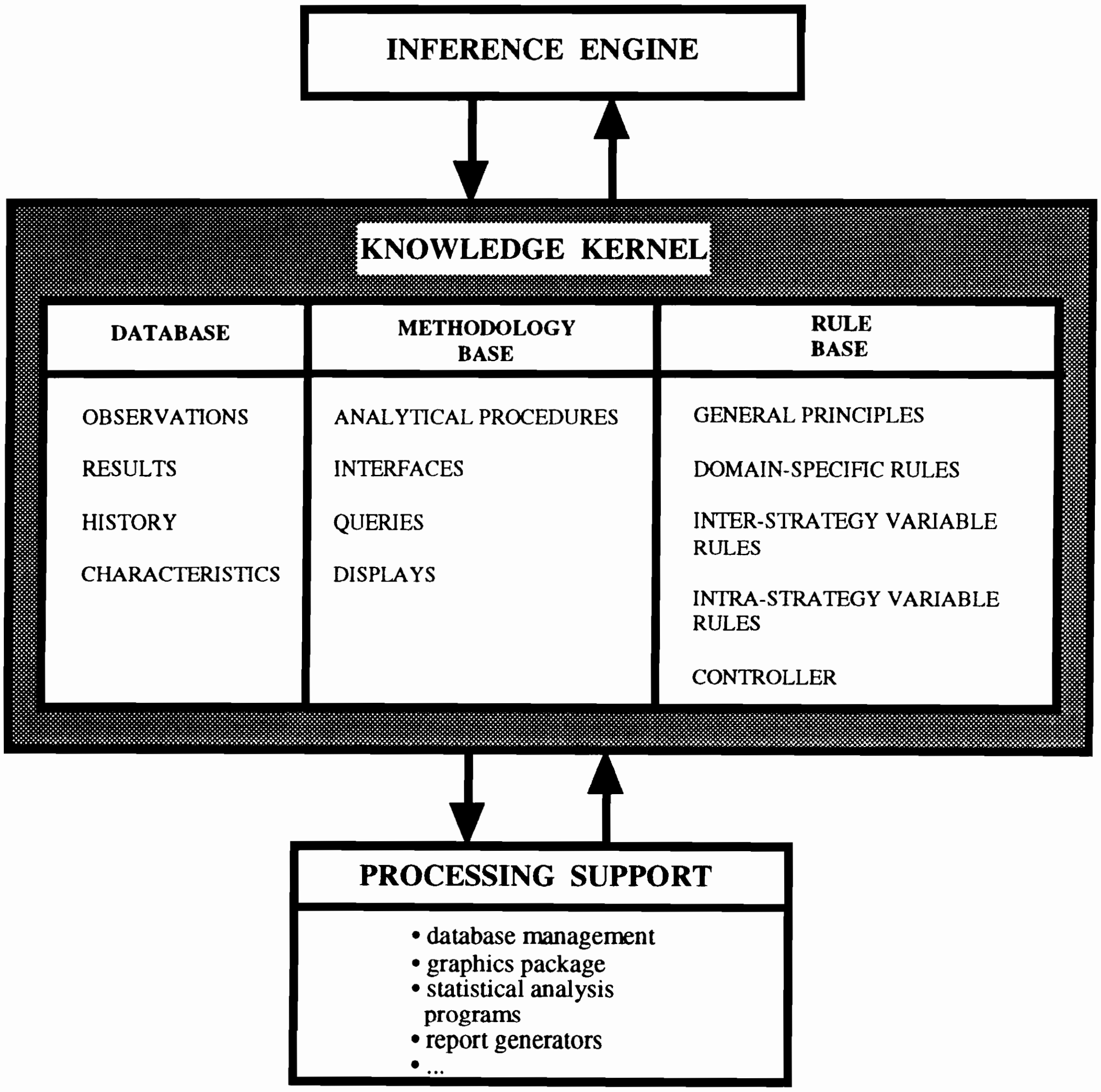


Figure 2.3: The analysis environment component (of figure 2.2) cast in a knowledge-based context

Database

The database portion of the knowledge kernel is used to store four types of information needed in the simulation optimization process: observations, results, history, and characteristics.

Figure 2.4 provides an example of how the first two types of information -- observations and results -- are related and how their data may be represented in the knowledge kernel. The information in figure 2.4 is structured using a relational data model approach. This type of diagram, where each data item is represented as an "object," is helpful in structuring both traditional relational databases and frame-based expert systems. The first type of information stored in the database, observations, includes the specific factor and response values for each simulation experiment. They are stored in the "settings" and "observations" data items in figure 2.4. The second type of information stored in the database, results, includes output from statistical analyses (regression slopes, ANOVA sum-of-squares, etc.). They are represented by the "regression" and "reg-parameters" data items in figure 2.4. The other data items in figure 2.4 contain additional details on each factor and response, as well as a record of which set of sample points was used in each analysis. (A single simulation experiment can be a part of several groups of observations or samples, and results from a single experiment can be used in more than one analysis.)

With the data stored in the manner shown in figure 2.4 and with the appropriate set of queries, one can easily determine such information as the estimated coefficients for a particular regression analysis, how well the model fit, what set of observations was used, and if necessary, the individual values for each of the factors that were used in the analysis.

The third database element, history, provides a longitudinal record of what operations were performed when. This record in conjunction with rules can be used to modify the strategy variables. For example, if the last two operations performed by the satisficer were gradient searches in the same direction, and each resulted in improvement in the sampled response, then this information

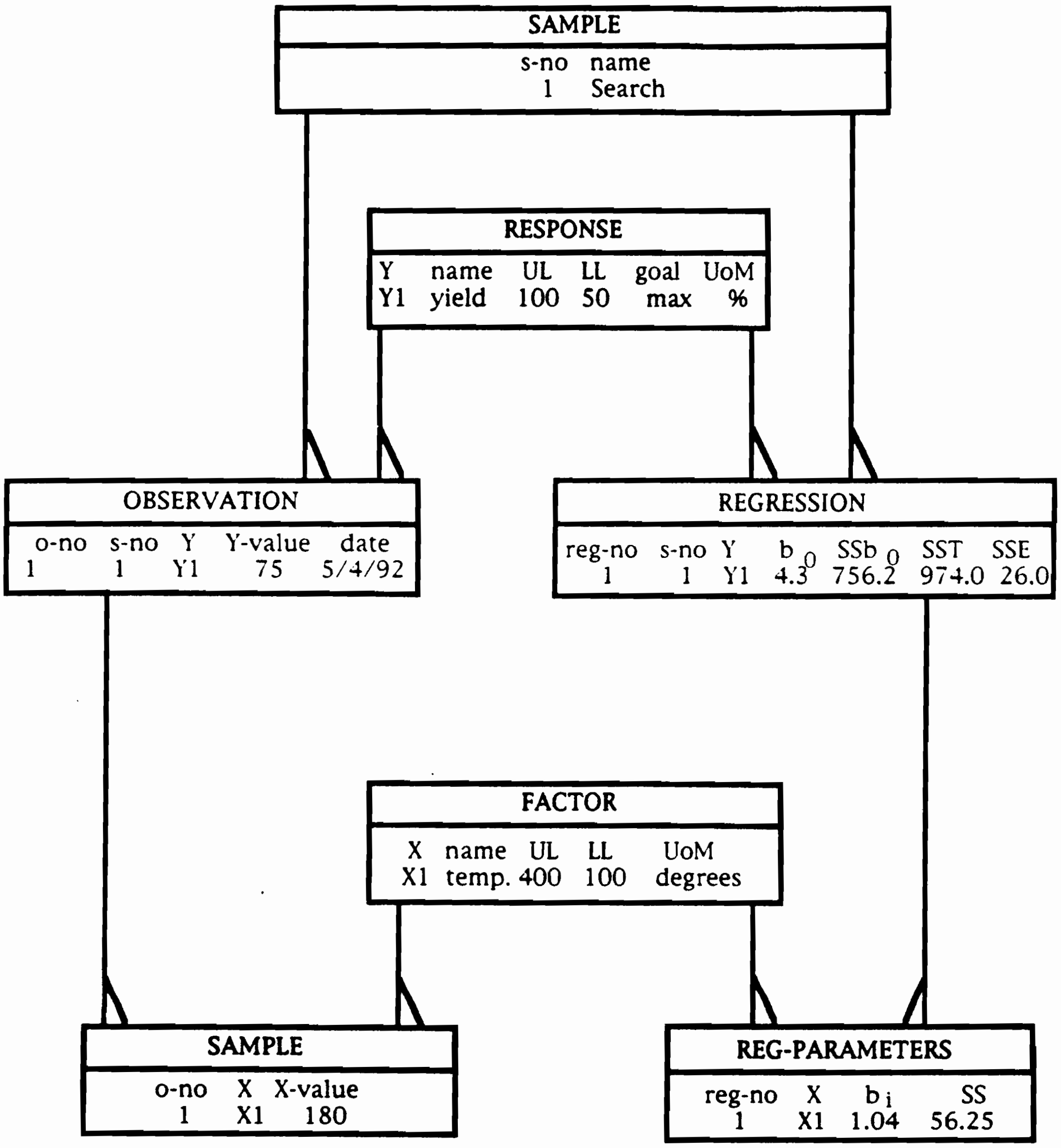


Figure 2.4: An example relational data model in the knowledge kernel

could be used to fire a rule that accelerates the search by increasing the step size. The historical information, in conjunction with the appropriate logic in the rule base, could also monitor the response function, characterize it (e.g., many local optima, constant variance), and build a history of its behavior.

The final database element, referred to as characteristic data, stores descriptive information on the various techniques used by the satisficer (e.g., how many observations are needed for a particular design, whether the design is first-order, rotatable, etc.). It could also include descriptions of known response functions. Periodically, this information could be compared to collected sample data, as described above, so that the response function under investigation could be classified in terms of shape (e.g., symmetric, ridge, flat), degree of variability, etc.

Rule base

The rule base portion of the knowledge kernel in figure 2.3 is subdivided like the database portion, but the elements of the rule base are more hierarchical, i.e., the lower-level rules are not fired until many of the upper-level rules have been processed. The manager or controller portion contains the lowest-level rule set. It is used for such management tasks as initializing the system when starting a new problem, re-starting the analysis of a problem that had been temporarily suspended, maintaining the history of operations, etc. It is also concerned with ensuring that the proper sequence of operations is followed, determining when it is necessary to collect data (pass control to the simulator), etc. For example, before a search operation is performed the controller should check to see that the factor values (i.e., values of decision variables) have been set based on a specified search direction and step size.

The next two elements in the rule base, listed above the controller in figure 2.3, involve the strategy variables. Inter-strategy variable rules are primarily concerned with the interplay between the strategy variables (for example, the relationship between the step size and the size of the design region or the type of search technique to use in conjunction with certain designs). Considering past operations is important as well, as illustrated by the example of an inter-strategy rule in figure 2.5. The example rule fires when the previous two operations had successful searches (resulted in improvements in the response) in the same direction. In this case, the firing of the rule causes the step size to double. It also causes a lower level rule in the hierarchy -- a controller rule -- to fire (as illustrated by the second rule in figure 2.5). This rule calls a procedure that determines the value of the factors based on the new step size (and prior direction) and a procedure to run the simulation model at this point and determine the response.

The other element in the rule base specifically related to the strategy variables, called intra-strategy rules, deals with expertise and guidance for a particular strategy variable. A few example rules include: if there is little variability in the response, then type of design is not that important; if response appears to possess a steep ridge, then a uniform-precision CCD design is preferred; if a simplex design was constructed, then use Box's complex search technique; if there is a drop in the response along a search path, then return to the prior sample point and set up another first-order design.

Domain-specific or problem-specific rules provide information from previous studies in comparable environments. For example, a response being measured in a specific type of production process may have been found in the past to be quite stable and exhibit relatively litle variability; or, a particular class of inventory problems may have been found in the past to possess a gently sloping, concave, and symmetric response function.

General principle rules are used to represent broader analysis knowledge, i.e., broader or more general concepts that apply to an entire class of techniques. For example, if a significant lack of fit is found in a first-order model, then it should be inferred that there is a need for a second-order model. Another set of general principle rules may provide guidelines for the removal of a factor from further consideration (i.e., factor screening).

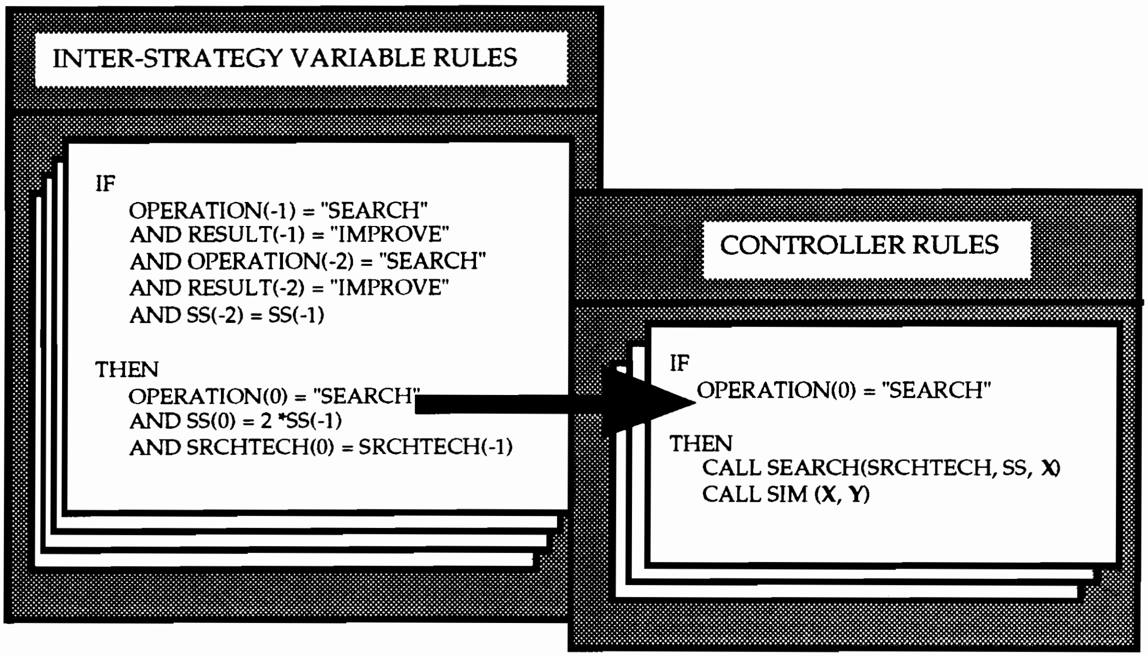

INTER-STRATEGY VARIABLE RULES
IF
OPERATION(-1) = "SEARCH"
AND RESULT(-1) = "IMPROVE"
AND OPERATION(-2) = "SEARCH"
AND RESULT(-2) = "IMPROVE"
AND SS(-2) = SS(-1)
THEN
OPERATION(0) = "SEARCH"
AND SS(0) = 2 *SS(-1)
AND SRCHTECH(0) = SRCHTECH(-1)
CONTROLLER RULES
IF
OPERATION(0) = "SEARCH"
THEN
CALL SEARCH(SRCHTECH, SS, X)
CALL SIM (X, Y)

Figure 2.5: Example rules in the knowledge kernel

Methodology base

The methodology-base portion of the knowledge kernel, as shown in figure 2.3, is composed of four primary elements: analytical procedures, interfaces, queries, and displays. The methodology base is considered at the same level in the framework as the database or rule base -- it should be just as easy to add a new statistical procedure or a new report to the methodology base as it is to add a new rule to the rule base or a new piece of information to the database.

The element referred to as analytical procedures includes all of the statistical tools needed in the simulation optimization process to compute such things as the regression parameters, analysis of variance, direction of steepest ascent, location of design points, etc. Implementation of these procedures is through processing-support software described earlier (programmed either in third-generation language subroutines or a part of a more general statistics package).

The interface element of the methodology base is a key factor in the design, development, and implementation of a simulation optimizer. Many different types of interfaces are needed to establish protocols and links between the many support routines and packages. A few example uses of interfaces follow. Obviously there must be a set of interfaces that allows the system to interact with a simulation model -- a separate interface is needed for each simulation language that is used by the satisficer so that a protocol is established for passing the factor values to the simulation program and receiving the response value(s) once the simulation experiment is complete. Another important set of interfaces links the expert system and the various computation routines. Well-defined user interfaces are of paramount importance to facilitate the interaction between the optimizer and users. Many screen displays are used for routine operations, but an optimizer also must provide "windows" for the user to directly interact with lower level processing environments, e.g., statistical analysis, database, graphic, etc. This is essential to support any type of ad hoc investigation and analysis.

Queries supply the satisficer with standard interrogations to the database or rule base. Database queries are critical to passing data to and from the statistical procedures. For example, an SQL-type query could provide access to data from a specified experiment, format it so that it can be read by a regression routine, receive output from the analysis, and store the results (coefficients, sum of squares, etc.) back into the appropriate place in the database. One type of rule-base query is a trace of the rationale for the optimizer recommending a particular operation.

Displays provide templates for the many graphical and report-format exhibits used in the simulation optimization process. For example, in RSM analyses many types of graphical displays are used. Contour plots are commonly used, but, given the advances being made in computer graphics there are many exciting opportunities for application of these concepts to support the RSM and simulation optimization analysis process. Many standard reports are used in the analysis process -- these range from a history of the search process to date (provides information on the region explored, improvement in the response, types of operations performed, etc.) to summary statistics resulting from the application of a regression procedure to a set of data.

Chapter 3: Use of a Classifier to Facilitate Simulation Optimization

Background

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 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 simulation model output (often referred to as the "response surface") provides an estimate of how the true system would respond to these inputs. For many systems the modeler wants to know the "best" possible set of inputs, in the sense that application of those inputs to the simulation model would result in highly desirable or even a "best" set of outputs. However, although simulation is very useful in predicting the system output or response, simulation does not in and of itself indicate the input conditions required to achieve the desired outcome. The process of finding the input conditions that will yield the optimal (or near optimal) system response is referred to as simulation optimization, and it can be a very expensive and time consuming activity.

Many different search strategies have been utilized or considered in simulation optimization, including response surface methodology (RSM), random search, single factor search, nonlinear programming, and some newer techniques such as simulated annealing and genetic algorithms. A critical decision that must be made in simulation optimization is which strategy to employ. Some work has been done to aid this decision, although, as Meketon (1987) concludes, "optimization for simulation, to date, remains an art, not a science." Meketon approached the problem of technique selection by considering the information available (or assumed) about the simulation itself and then grouping the optimization methods accordingly. Safizadeh (1990) discussed a variety of strategies and their application. He concluded that, in general, RSM approaches are most effective, although some of the newer developments look promising. Smith (1973) performed an empirical study of the effectiveness of several search strategies (random search, two variations of 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.).

The current state of the literature suggests that organized guidance is needed to help users choose appropriate search strategies. Safizadeh explains that "for successful design and analysis of simulation, one should be well versed in several disciplines." Due to shortcomings along these lines, users are inhibited from using simulation optimization (and hence, sometimes, simulation). He concludes that there is, therefore, a need to "develop interactive programs which direct a user to an appropriate optimization technique."

In Greenwood, Rees, and Crouch (to appear), we have suggested that part of the reason for the need for such a program to direct users is that simulation optimization is an intermingled combination of "art" and "science." Although there are clearly-defined algorithmic procedures, there are also rules of thumb, which vary from situation to situation. For example, different approaches in setting step size during the search are discussed by Myers (1971) and Biles (1975); deciding when to stop searching in a specified direction is evaluated by Myers and Khuri (1979); Daughety and Turnquist (1978 and 1981) present a novel "budget-focused" search procedure that considers the allocation of simulation runs both temporally and spatially. Our solution to the art/science problem was to suggest separating this art and science in simulation optimization, by storing heuristics for the different search procedures in a knowledge-based system and the science or algorithmic portion in a so-called methodology base. The heuristics can then select the most appropriate search strategy for the occasion and subsequently invoke procedural aspects for that strategy as required. In this manner, rules of thumb as well as new procedural modules can be added, updated, and deleted without destroying a fixed "algorithm." The "algorithm" is embedded in the (knowledge base) rules, which are easily changed.

In the work referenced above (Greenwood et al.) we also specified a broad-brush architecture for a knowledge-based simulation optimization system. However, we had not actually built such a system at that time and hence were not able to specify implementation considerations. Having now built a system for small problems, we are able at this point to stipulate a precise and improved architecture and process flow for this knowledge-based system. This is the contribution of this research. Moreover, a major design decision made in the present research was to put a Classifier module at the heart of the system. As will be explained, the Classifier categorizes response surfaces into classes, where each class consists of those surfaces most amenable to a given search technique.

The remainder of this chapter is organized as follows. In the next section, the specific architecture of the Classifier KBSOS is presented. Following that section, the classifier KBSOS process flow is explained, with details of the classifier and the knowledge base revealed. Third, the (computer) solution of an example problem is provided to illustrate the system and to demonstrate advantages of the classifier KBSOS over single-technique approaches. Finally, conclusions and future work are discussed.

The Classifier KBSOS Architecture

The first detailed notion of a knowledge-based simulation optimization system (KBSOS) is presented in Greenwood, Rees, and Crouch (to appear) and is shown in figure 3.1. As may be seen from that figure, the major elements of that system are an inference engine, a so-called knowledge kernel, and processing support. The inference engine, as is standard in all expert-system shells, provides inference and control capability and operates on knowledge stored in the knowledge base in the knowledge kernel. The processing support component provides database management, graphics packages, statistical analysis programs, report generators, etc. As explained in Greenwood et al., the knowledge kernel is the hub of the system, and itself consists of three major modules: the Data Base, the Methodology Base, and the Rule Base. The Data Base contains historical information: records of decisions made in each simulation optimization and the results of those decisions. Moreover, this module stores what is currently being experienced during each round of simulation optimization; this includes the data generated and decisions made during strategy planning, simulation results obtained during the ensuing search, regression coefficients, analysis of variance (ANOVA) results for lack-of-fit tests, etc. The Methodology Base includes analytical procedures for statistical computations, algorithms to conduct the "science" portions of the various search algorithms available in the KBSOS, and requisite interfaces, queries, and displays. As proposed in Greenwood et al., the statistical components include submodules to do specialized regression, ANOVA, steepest ascent, canonical analysis, experimental design, etc. The final component of the knowledge kernel is the Rule Base. As shown in figure 3.1, the Rule Base includes rules for general principles, domain-specific rules, inter-strategy rules, intra-strategy rules, and a controller. But although Greenwood et al. allowed for strategy selection in the Rule Base, no prescription was given for how to do it, i.e., determine which search technique should be applied to a given surface. That research merely specified that rules be used to select a strategy. Moreover, there was no overall organized scheme elucidated for determining this mapping. It is these major "gaps" or limitations that are now addressed as we specify our new architecture.

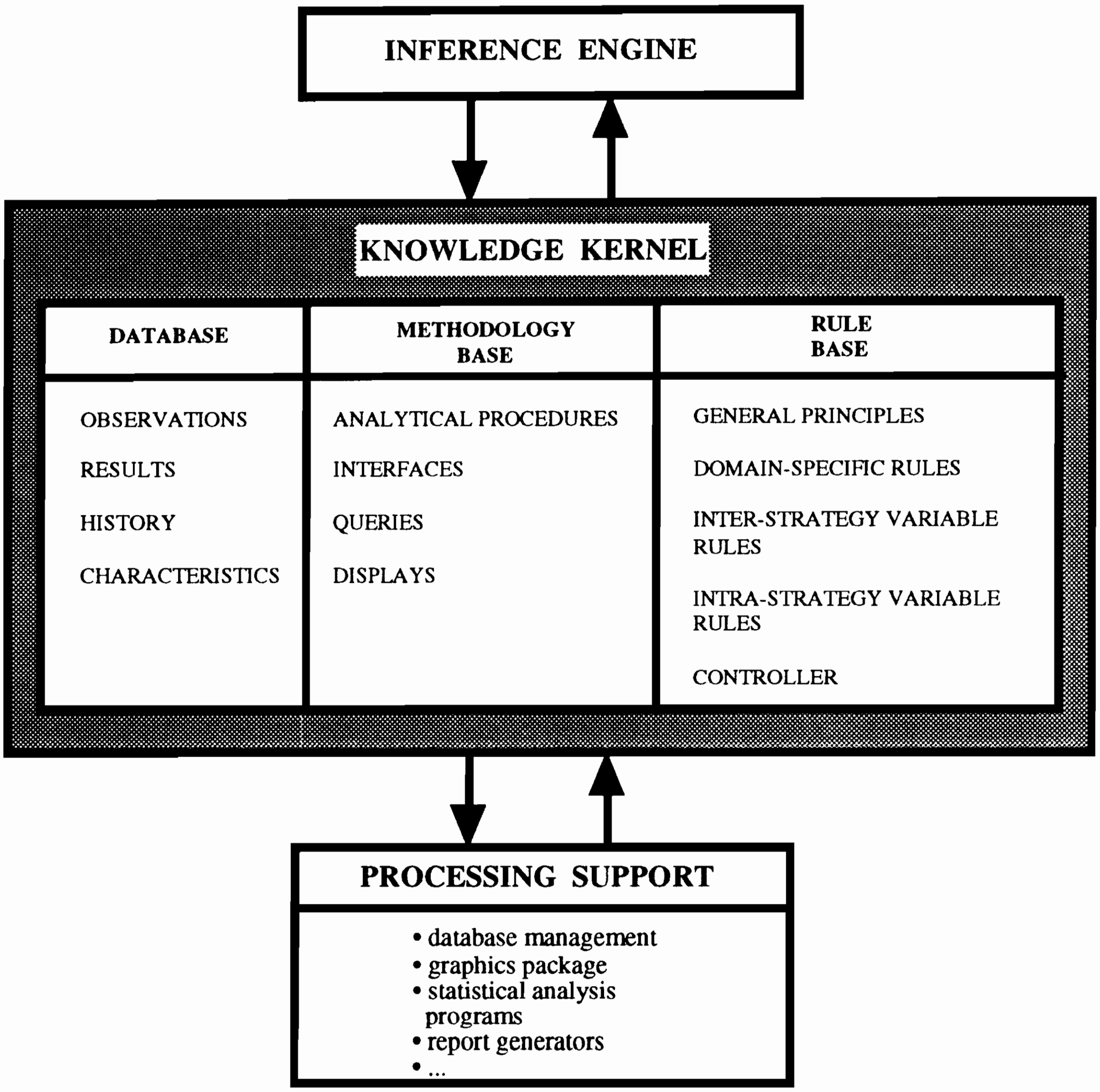


Figure 3.1: Greenwood-Rees-Crouch simulation optimization architecture

In order to develop an architecture that goes beyond Greenwood et al. and specifies in greater detail how heuristics in the rule base can guide simulation optimization, we observe the following points. First note the obvious fact that different simulation models generate different response surfaces. Second, observe that different search strategies (e.g., RSM, single factor search, etc.) work with different degrees of success on different surfaces. In fact, the literature contains many such findings (see, for example, Meketon (1987), Safizadeh (1990), and Smith (1973)), although these results frequently are neither published in one central location (say under "simulation optimization"), nor are they identified per se as rules of thumb to be used in simulation optimization. Next, and most critically, note that it would be highly desirable to classify any given response surface according to that search technique which has the highest likelihood of generating the best solution. For example, a given surface with many local optima might be placed in the "simulated annealing" search strategy category, whereas a simpler surface with a global optimum might be categorized in the "RSM" category, because those search strategies are deemed most appropriate and likely to generate success. Stating the categorization idea more succinctly, we postulate that response surfaces can be categorized using a Classifier according to those characteristics that make various search techniques apropos.

Determining those characteristics that partition surfaces into search categories is itself a major research question. Smith, in his 1973 ***Operations Research*** article, laid the groundwork and demonstrated feasibility of this approach. He examined different surfaces on six factors: presence of local optima, size of random error, distance of the starting point from the true optimum, the number of controllable factors, the number of available computer runs, and the relative activity of the controllable factors. He found that different search techniques performed with varying success on response surfaces at the different levels of his six factors. For the research reported here, we adopt Smith's six factors as a starting point and remark that additional characteristics described in the literature as significant can be added at a later date. We also note that this mapping from response surface to most appropriate search technique is best incorporated in a knowledge base, as the mapping is likely to be described as rules of thumb rather than fixed algorithmic procedures.

Once classification of surfaces by search technique has been determined, then the appropriate search technique with its heuristic and procedural components can be invoked and simulation runs can be made. With additional data points from these runs, the classification can be redone, because the shape of the response surface in the newer search region may suggest a different search technique. The process repeats until the simulation optimization is completed.

A simple representation of the simulation optimization system described above is shown in figure 3.2. Note that in this figure the optimizer is shown in a feedback path around the simulation model. With this configuration, the optimizer suggests improved inputs to the simulation model, based on outputs the optimizer has obtained from the simulation. Moreover, the optimizer itself consists of two parts: a classifier (to categorize the surface) and a set of heuristics to specify the search technique. In particular, when a set of inputs is presented to the simulation model, the model generates points on a corresponding output surface. These, in turn, are used to ***classify*** (note the "C" in figure 3.2) the surface as best as possible as to several general criteria or characteristics, as mentioned above. Those criteria are then fed as inputs to the ***heuristic*** knowledge base ("H" in figure 3.2), which recommends a most-preferred strategy for the given information on the surface. Implementation of this search strategy results in a new set of inputs that the simulation optimization system thinks will lead to a yet more desirable set of outputs. The process repeats for many iterations until a sufficiently desirable output is obtained. As indicated, with the proposed system it is possible for the classifier to change search strategies as different portions of the response surface are categorized and explored. This is a powerful feature of the classifier model.

With the Classifier approach to simulation optimization outlined, it is now possible to specify a new architecture for the optimizer module. In actuality, the new architecture (figure 3.3) is very similar to the Greenwood et al. architecture shown in figure 3.1. The newer architecture contains the same three major portions, the inference engine, a processing-support module, and a knowledge kernel. The latter still consists of three modules, a Data Base, a Methodology Base, and a Rule Base. But now that we have actually built the Rule Base, we have found it makes more sense to redefine it as follows. There are still five different rule sets in the new Rule Base, but some differ

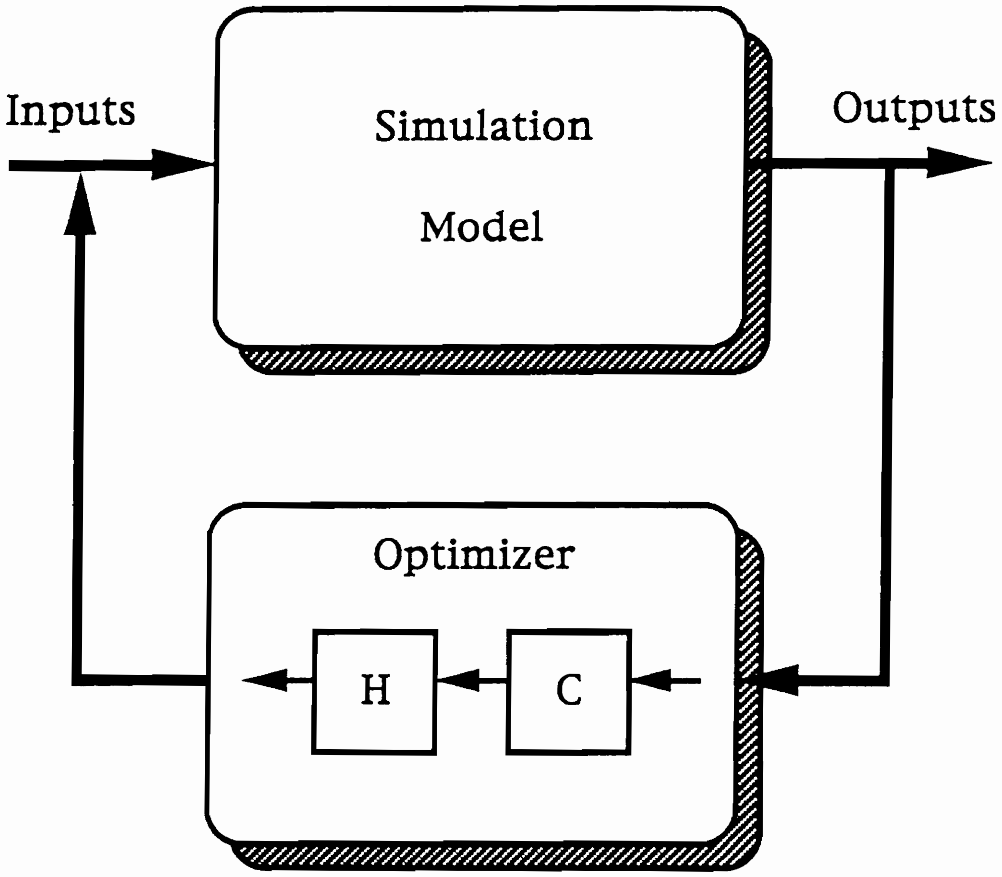


Figure 3.2: A simulation optimizer with a classifier and heuristics

from the earlier proposal. As before, there is a Controller set of rules, but now the Controller rules sit hierarchically above the other four rule sets, calling each as appropriate. As shown in figure 3.3, these four rule sets are User Rules, Classifier Rules, Strategy Selection Rules, and Strategy Detailer Rules. The first rule set is used to transform user inputs into terms usable by the rest of the rule sets. (For example, such simple rules as "IF num_inputs < 4 THEN num_factors = small" translate user-supplied information into a form usable by the strategy selector, as will be explained later.) The second rule set, the Classifier Rules, contains the heuristics necessary to run the Classifier and categorize surfaces. The last two rule sets are the same as we earlier suggested in Greenwood et al., but we have changed the names. What was inter-strategy variables rules is now Strategy Selector rules. They are equivalent in the sense that it is the rules that "cut across" strategies that specify which strategy to select. Similarly, the Strategy Detailer rules are a renamed version of the intra-strategy variable rules. That is, once a given strategy is chosen by the Strategy Selector rules, then implementation details for that strategy can be specified by what used to be called intra-strategy rules, but now are called Strategy Detailer rules. Finally note by comparing figures 3.1 and 3.3 that we no longer separate out architecturally the "general principles" and "domain-specific" rules; rather we incorporate each of these as appropriate in the other rule sets.

Having now defined an architecture for a Classifier Simulation Optimization system, we turn our attention to the flow or process such a system would follow.

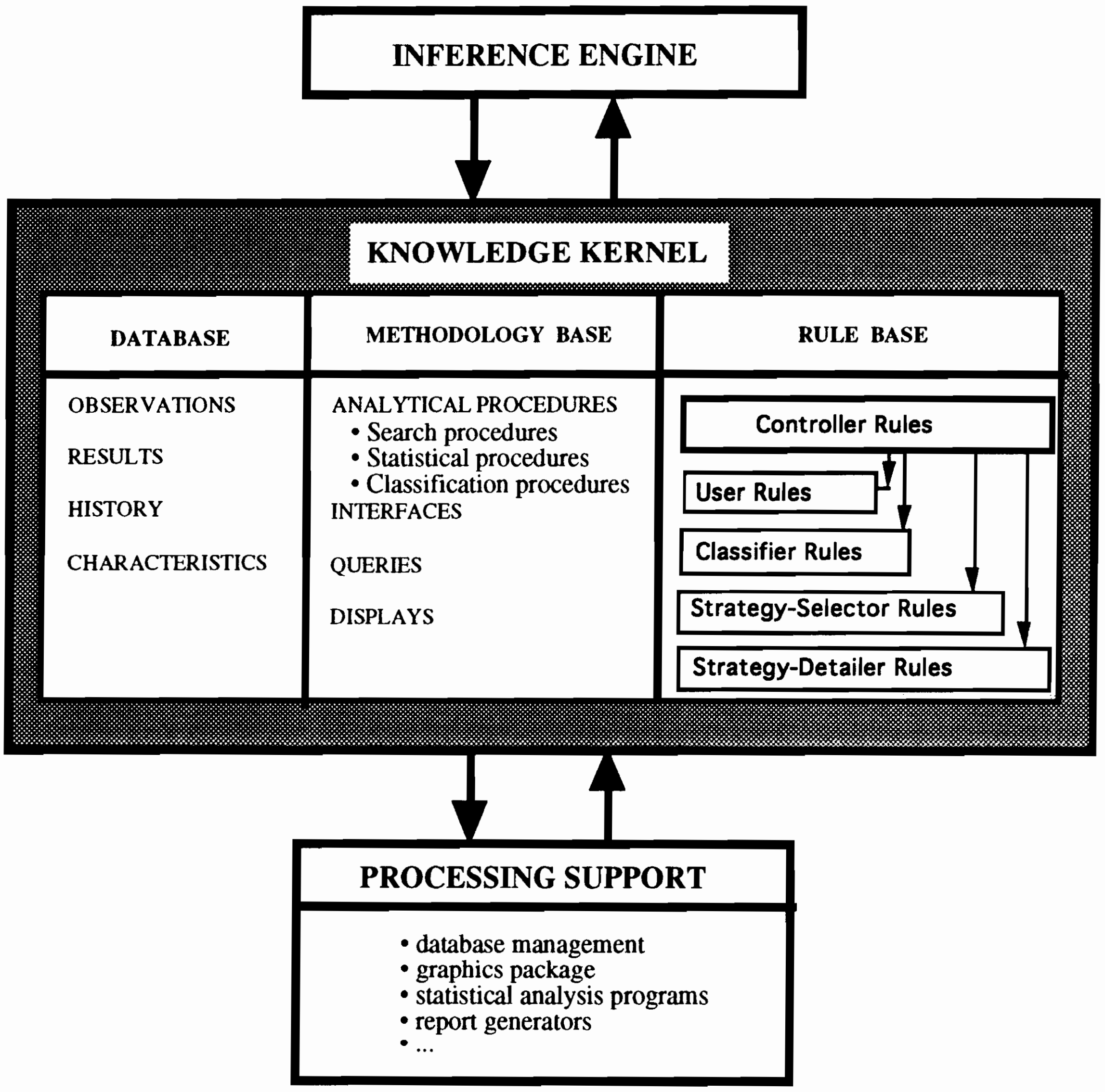


Figure 3.3: The Classifier knowledge-based simulation optimization architecture

The Classifier Process Flow

Overview

In figure 3.2 we defined the basic process flow of our Classifier simulation optimization system. There we showed that there are three basic steps in optimization: classify the surface (the "C" in the optimizer box), use heuristics to recommend a search strategy for that surface (the "H" in the same box), and then search with the simulation model. The process interates until optimization is complete.

Figure 3.4 shows a somewhat more detailed view of the Classifier KBSOS process flow, including user input and breaking down the heuristic portion into two separate functions. In particular, the **USER** module elicits information from the user about the problem, termination conditions, etc. The **CLASSIFIER** module processes user information as well as data from any relevant previous searches and characterizes the simulation response surface as to several key variables. The **STRATEGY SELECTOR** operates on input from the Classifier and User modules and selects a search strategy. Information from the User, Classifier, and Strategy Selector modules is then processed by the **STRATEGY DETAILER**, which specifies operational details necessary to perform the **SEARCH** stipulated by the Strategy Selector.

Recall that when we mention the word "module" in connection with any of the five boxes referred to above, we are not describing a procedural algorithm that is called and executed. Rather, as has been explained and will be illustrated in the example below, a set of rules is invoked that in turn makes calls to only those procedures required by the particular rules that fire.

We now explain the the inner workings of the Classifier KBSOS flow in detail, component by component. Figure 3.5, a yet more detailed version of figure 3.4, is utilized for this purpose.

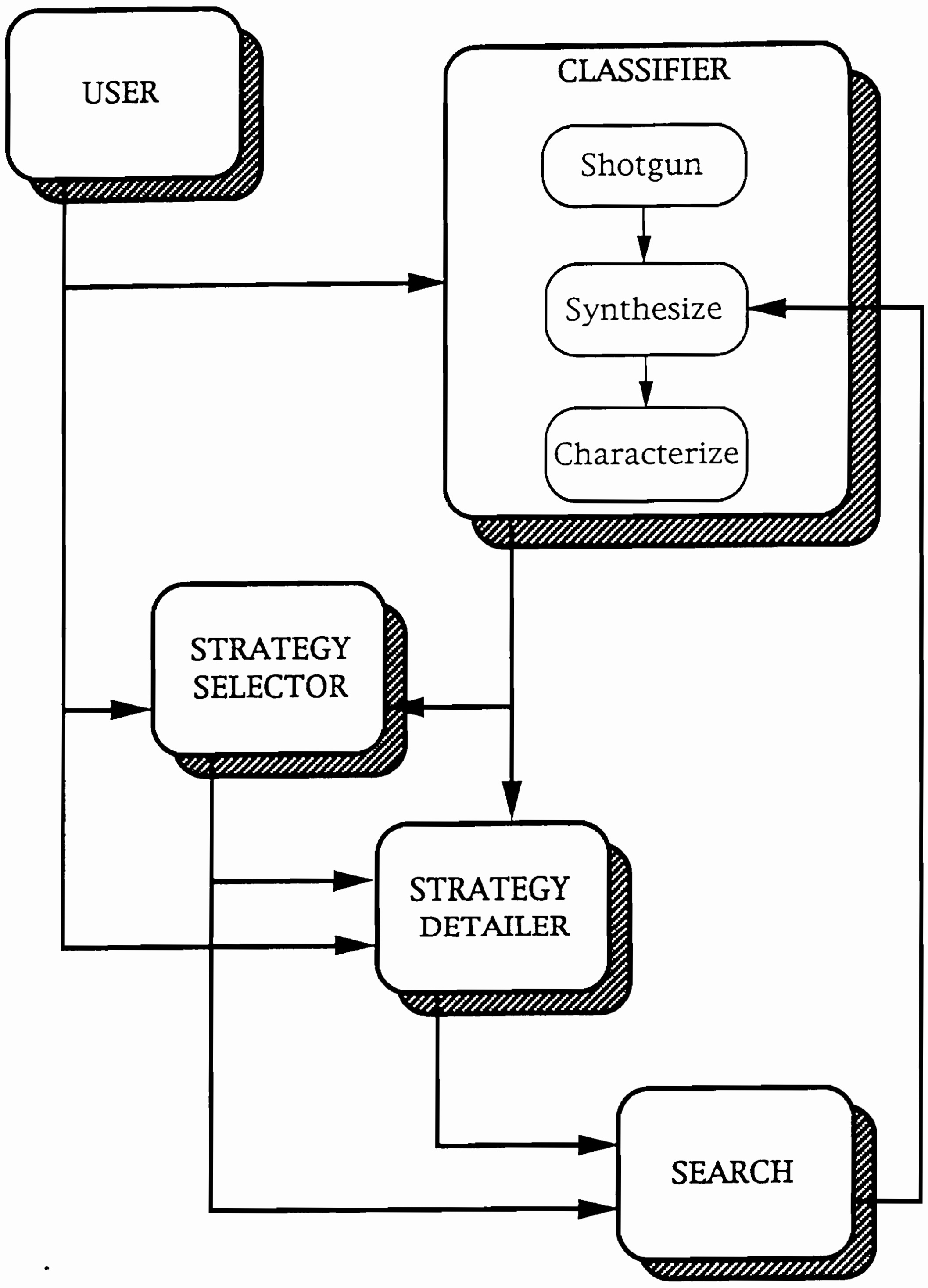


Figure 3.4: Process flow of the Classifier KBSOS

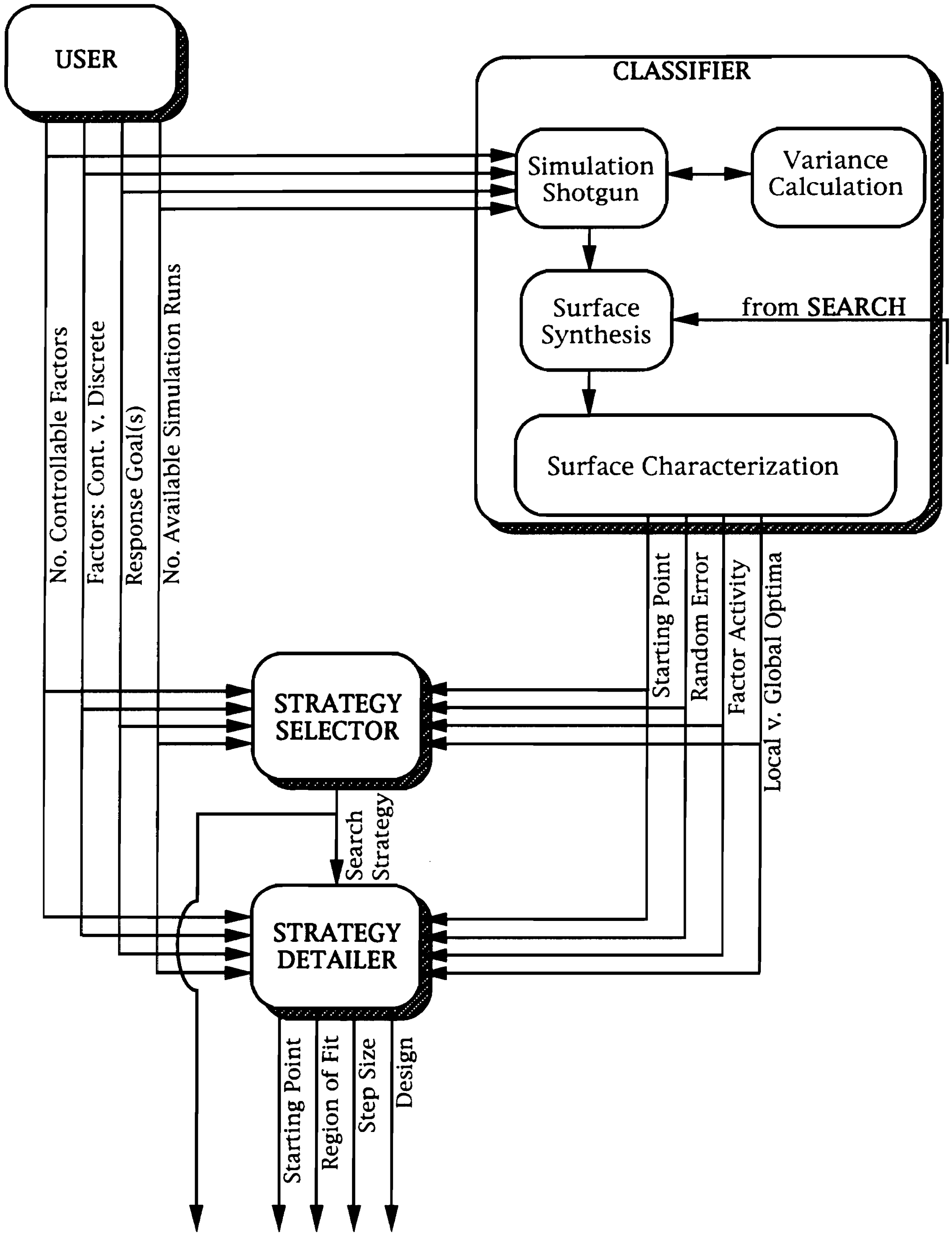


Figure 3.5: A detailed view of the Classifier KBSOS

USER module

The first step in the optimization process is to obtain from the user a description of the problem to be solved. This includes a specification of the number of inputs and outputs that the simulation has, a listing of the input factors (e.g., the domain of definition for each variable and whether variables are continuous or discrete), and an indication of which of these factors are controllable, and hence are to be adjusted by the optimizer, and which factors are not. Response goals, one for each simulation output, are also set. For example, it might be stipulated that for output response y_1, a minimum yield of 85% must be reached (i.e., $y_1 > 0.85$). Moreover, data as to limits on computer resources and termination conditions are also recorded. This may be specified in several different manners, including limits on execution time, number of available computer runs, etc.

User specifications must in some cases be translated from language in which the user expresses concepts, to those terms needed by the other modules in the KBSOS. This is a second function of the User module, and it is easily implemented with simple rules.

CLASSIFIER module

As mentioned, Meketon (1987), Safizadeh (1990), and Smith (1973) have all explicitly examined the search-technique selection problem in the simulation-optimization context. Smith's work, published in ***Operations Research,*** is the most complete for our purposes and consists of an empirical investigation of seven different search strategies. He developed a full-factorial experimental design, testing the seven search strategies under six different simulation scenarios. The seven strategies are random search, single factor search, single factor search - accelerated, and four different RSM variations.

Each of the simulation scenarios he investigated consisted of one of the 128 possible combinations of the following six factors at the levels indicated in parentheses:

1. local optima (present or absent)
2. size of random error (small or large)
3. distance of the starting point from the true optimum (near or far)
4. the number of controllable factors ($= k = 30$ or 120)
5. the number of available computer runs (0.5k, 1.1k, 1.5k, or 2.0k), and
6. the relative activity of the controllable factors (low or high).

The measure of success of a given search strategy under a specific case in Smith's experiment was indicated by a number between 0 and 1 inclusive representing the fraction achieved of the total possible improvement (i.e., the difference between the values of the response surface at the starting and optimal points).

As mentioned, what Smith discovered in his experimentation is that the choice of best search technique varies significantly with the factors he explored. That is, an optimal search strategy can only be determined as the factors local optima present/absent, size of random error, etc., are known. This suggests, as we have postulated, that different search techniques should be selected based on the problem at hand.

Since Smith has posited the above six factors as affecting search technique and has found significance, and since it "makes sense" to us intuitively that these factors would influence choice of strategy, we adopt his six factors as the discriminating variables in our surface classifier. There is no claim here that these six factors are the only and ultimate influences upon strategy selection; our assertion is that the selection of these six items is a reasonable starting point for this research.

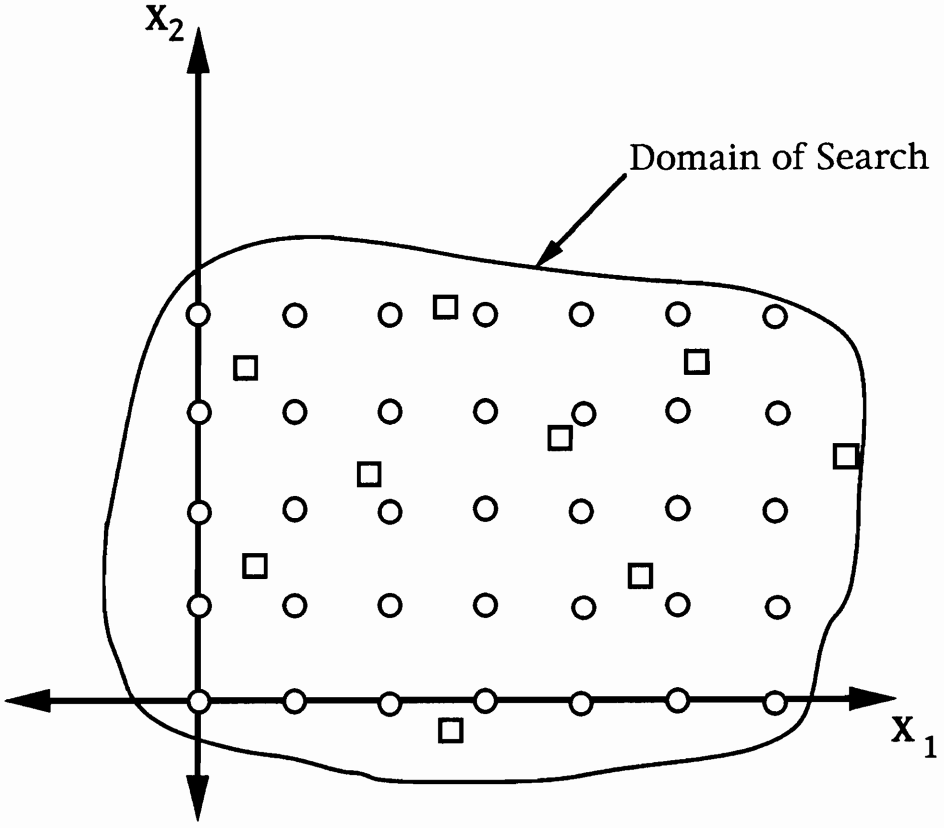
It will be noted from figure 3.5 that the classifier has four outputs; these outputs are Smith's first, second, third, and sixth factors. These four factors will be estimated in the Classifier and passed on to the rest of the optimization system. The other two of Smith's factors (numbers four and five) are specified by the user in our model, and hence are shown as emanating from that box in figure 3.5.

The purpose of the Classifier, therefore, is to estimate the four output characteristics shown coming from ***that*** box in figure 3.5. Generally speaking, this is done in three sequential steps:

- "ordering" the simulation to make a prescribed number of simulation runs
- fitting a surface through these points (i.e., simulation outputs), and
- classifying the fitted surface according to Smith's four output characteristics.

Four modules are used within the Classifier to do these tasks. The first two modules, the *Simulation Shotgun* and the *Variance Calculator,* specify the number and location of simulation runs. The *Shotgun,* so named because it scatters runs across the input space, uses heuristics and the user-specified upper limit on simulation runs to determine the number of runs to make. Most simulation runs are made at regular intervals throughout the input space. This can be visualized as a "grid" of runs over the domain, as indicated in figure 3.6 by the circle-shaped points. (These runs are, of course, made at discrete levels for all non-continuous variables.) In case there are variations in the response surface at the same wavelength as the spacing of the grid points, some runs are also made at randomly-chosen input levels (the square-shaped points in figure 3.6). The *Variance Calculator* is used to suggest whether the surface is highly volatile and hence needs extra simulation runs over the region. It also provides data to the *Surface Characterization* module as to the amount of noise present in the surface.

The second step in the Classifier procedure is fitting a surface through the responses gathered from the simulation runs made in step one. Currently a neural network is used to do this, although any successful functional synthesis approach may be used. All available simulation runs are used to train the neural network. The network is then used to estimate the responses for input levels between those of the actual simulation runs. This saves the time and expense of performing a large number of simulation runs.



○ indicates grid points over the domain of search

□ indicates random points over the domain of search

Figure 3.6: Simulation runs suggested by the shotgun

The particular approach we are currently using with respect to the neural network implementation is a backpropagation network trained on an orthogonal polynomial representation of the input data. (See Pao (1989) for the benefits and details of this.) That is, we use as inputs to the neural network the input variables themselves, a cosine and sine term of each input variable, three harmonics of each of these four terms, and five cross-product terms. For example, for a two-input variable (x_1, x_2) problem, there would be twenty-three inputs to the neural network, namely:

$$x_1,\ x_2;\ \cos(n\pi x_1),\ \sin(n\pi x_1);\ \cos(n\pi x_2),\ \sin(n\pi x_2);\ n\ =\ 1,2,3,4;$$

$$x_1 x_2;\ x_1\cos\ (\pi x_2),\ x_1\sin(\pi x_2),\ x_2\cos(\pi x_1),\ x_2\sin(\pi x_1).$$

Although further research is needed to determine the best procedures for implementing functional synthesis, we have had good success with the implementation described above.

The final step in the Classifier algorithm is determining values for Smith's four surface characteristics. This is done with procedural (third-generation) modules and heuristics (rules), all operating upon the synthesized surface and the variance calculations. For example, one such module determines whether local or rather global optima are present. To illustrate the workings of this module consider the simple case of a single input variable to a simulation program (call it x), that in turn would generate a true single-response surface (call it y) if all values of x could be run through the model. Since only a few values of x are suggested by the *Shotgun* to be run through the model (in order to conserve runs), only those points are known to be on the surface y. Hence an estimate of the surface y must be made, which is the function of the (neural network) surface synthesizer. Assume that the curve $\hat{y}$ shown in figure 3.7 is obtained by the network. Now the local optima present/absent module may be invoked, which works as follows. By search over the fitted surface, the highest value(s) of $\hat{y}$ is (are) found. In the example of figure 3.7, only a single highest point would be found at this step (call it $\hat{y}_{opt}$). Using a predetermined increment size δ (as indicated in figure 3.7) a search is made over the fitted surface $\hat{y}$ to see how many x exist at the value $\hat{y}_{opt} - \delta$. By continuing to search over x for values of $\hat{y}_{opt}$ - $k\delta$, where k is a positive integer, it is possible to see whether only a single optimum exists, or whether many optima appear to be present. This is indicated in the figure, where at $\hat{y}_{opt}$ - 3δ, two additional optima are observed.

STRATEGY SELECTOR module

The Strategy Selector consists of rules that take into account information from the User module and judgments made by the Classifier in order to choose the most appropriate search strategy. This rule set has been developed from the results of the experiments performed by Smith (1973) described above. The 128 data points presented in his tabular results were first converted to 128 rules. These 128 rules were then processed by a Generalization module in a learner we built (see Greenwood et al., forthcoming) and were therein reduced to 50 more general rules by that learner.

The rule antecedents (i.e., the **IF** parts) of the strategy selector rules correspond to the six surface characteristics Smith used to categorize simulation situations. The rules' consequent element (i.e., the **THEN** part), search strategy, can take on one of the following values: random search, single factor, single factor - accelerated, response surface methodology (RSM) I, RSM I - accelerated, RSM II, or RSM II - accelerated. The strategy chosen is the one which provided (for Smith) the maximum gain for the given surface characteristic combination (i.e., antecedent). Thus the Strategy Selector uses the User module's interpretation of user-supplied parameters and the Classifier's determinations of surface characteristics to decide which search strategy is expected to get closest to the optimum. A few of the (VP-Expert) rules in the Strategy Selector are shown in table 3.1.

STRATEGY DETAILER module

In order to implement the chosen search strategy, the values of a number of strategy variables must be set. This function is performed by the Strategy Detailer, which employs information from the

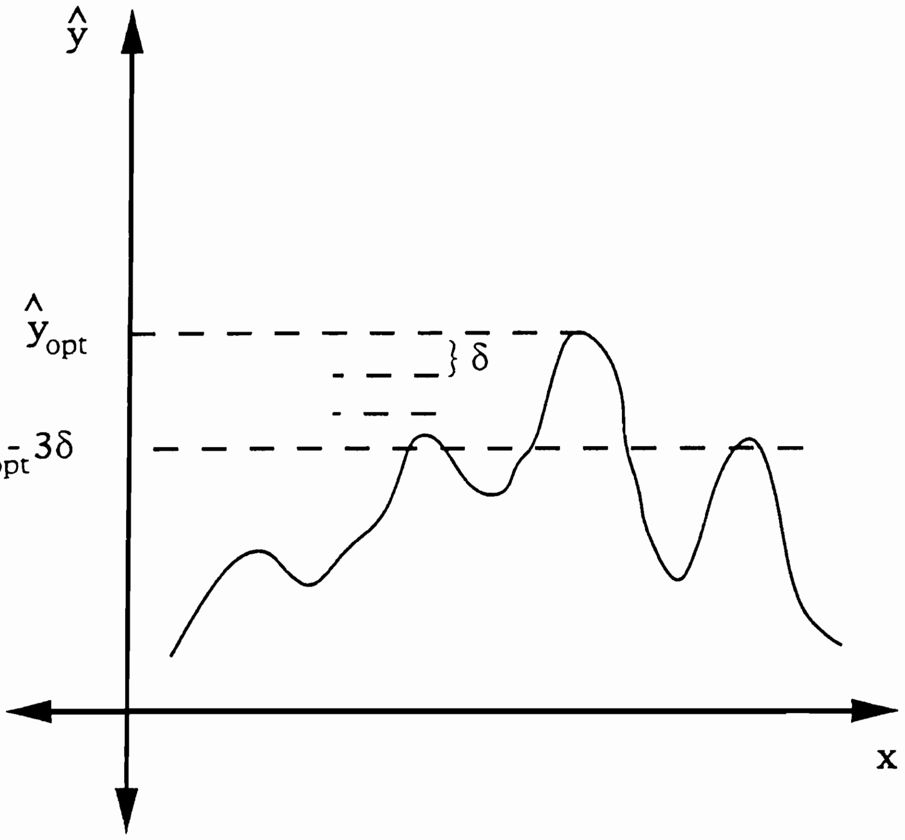


Figure 3.7: Determination of global vs. local optima

Table 3.1: Example strategy selector rules

```
RULE 11
IF        num_factors = small      AND
          local_optima = present   AND
          factor_activity = high
THEN      search_strategy = random_search;


RULE 19
IF        num_factors = small      AND
          num_sim_runs = small     AND
          local_optima = present   AND
          dist_to_opt = near       AND
          random_error = large     AND
          factor_activity = low
THEN      search_strategy = single_factor;


RULE 21
IF        num_sim_runs = medium    AND
          local_optima = absent    AND
          random_error = large     AND
          factor_activity = high
THEN      search_strategy = RSM_I;


RULE 32
IF        num_factors = small      AND
      num_sim_runs = large     AND
          local_optima = absent    AND
          dist_to_opt = far        AND
          random_error = small
THEN      search_strategy = RSM_I_accel;


RULE 48
IF        num_factors = large      AND
          num_sim_runs = small     AND
          local_optima = absent    AND
          random_error = small     AND
          factor_activity = high
THEN      search_strategy = RSM_II_accel;
```

User, Classifier, and the Strategy Selector. The Strategy Detailer component must decide several incidentals, including:

- where to begin the search (i.e., starting point)
- what will be the region of fit/search
- what design will be used, and
- the value of the step size.

Whether all of these are relevant depends on the strategy selected; in a random search, for example, starting point, design and step size do not come into play.

The Strategy Detailer knowledge base (KB) loads in the search strategy chosen by the Strategy Selector KB, and then uses this to fire the relevant detailing rule(s). One Strategy Detailer rule (as written in VP-Expert) is:

```
RULE rand_srch
IF  search_strategy = random_search
THEN starting_point = none
    CALL REGION,""
    step_size = none
    design = none;
```

"REGION" is a procedural program stored in the Methodology Base that uses information generated during classification to set boundaries for the region in which the search will be performed.

SEARCH module

After decisions have been taken as to search strategy and the strategy variables, the appropriate procedures in the Methodology Base are called on to carry out the chosen strategy. After this strategy has been pursued, the newly acquired simulation results are fed back to the Classifier's surface synthesizer (neural network). The neural network is retrained to include this new information, and the decision process repeats as described above from this point. Thus as more information about the response surface is obtained, the KBSOS reevaluates the most appropriate search strategy with which to continue.

Example

The purpose of this example is to illustrate the Classifier KBSOS algorithm and to show the benefits of the Classifier approach. To accomplish these objectives, an example is chosen that consists of a known surface plus random error; with this approach, it is possible to follow easily the progress of the search toward the optimum. As described in the preceding section, the Classifier KBSOS algorithm has been implemented using the expert system shell VP-Expert (1989).

The simulation model is described by the following equations (expressed in polar coordinates):

$$y(r,\theta) = 0.5 \exp(-2.7r^2) \cos(3\pi r) + 0.5 + \varepsilon, \ \ r \leq 0.5$$
$$y(r,\theta) = 0.5 \exp(-2.7r^2) \cos(3\pi r) \cos(4\theta) + 0.5 + \varepsilon, \ \ r > 0.5$$

where epsilon is $N(0,0.05^2)$. A three-dimensional plot of the response surface y (exclusive of error term) is shown in figure 3.8, and a contour plot of the surface is indicated in figure 3.9. Note that y is contained in the closed interval [0,1]. For ease of exposition, we will assume that the user, who is unaware of the surface shape or its optimum, thinks of the problem in a Cartesian-coordinate system. We take the origin of our system to be the (x_1, x_2) point at which the highest point on the curve $(y = 1)$ occurs.

Activity begins with the Controller (see figure 3.3), which is implemented in VP-Expert with ACTIONS blocks. Basically, the Controller "chains" (i.e., passes control) from one rule set to the next, requesting specific information from each, as will be indicated in the sequel.

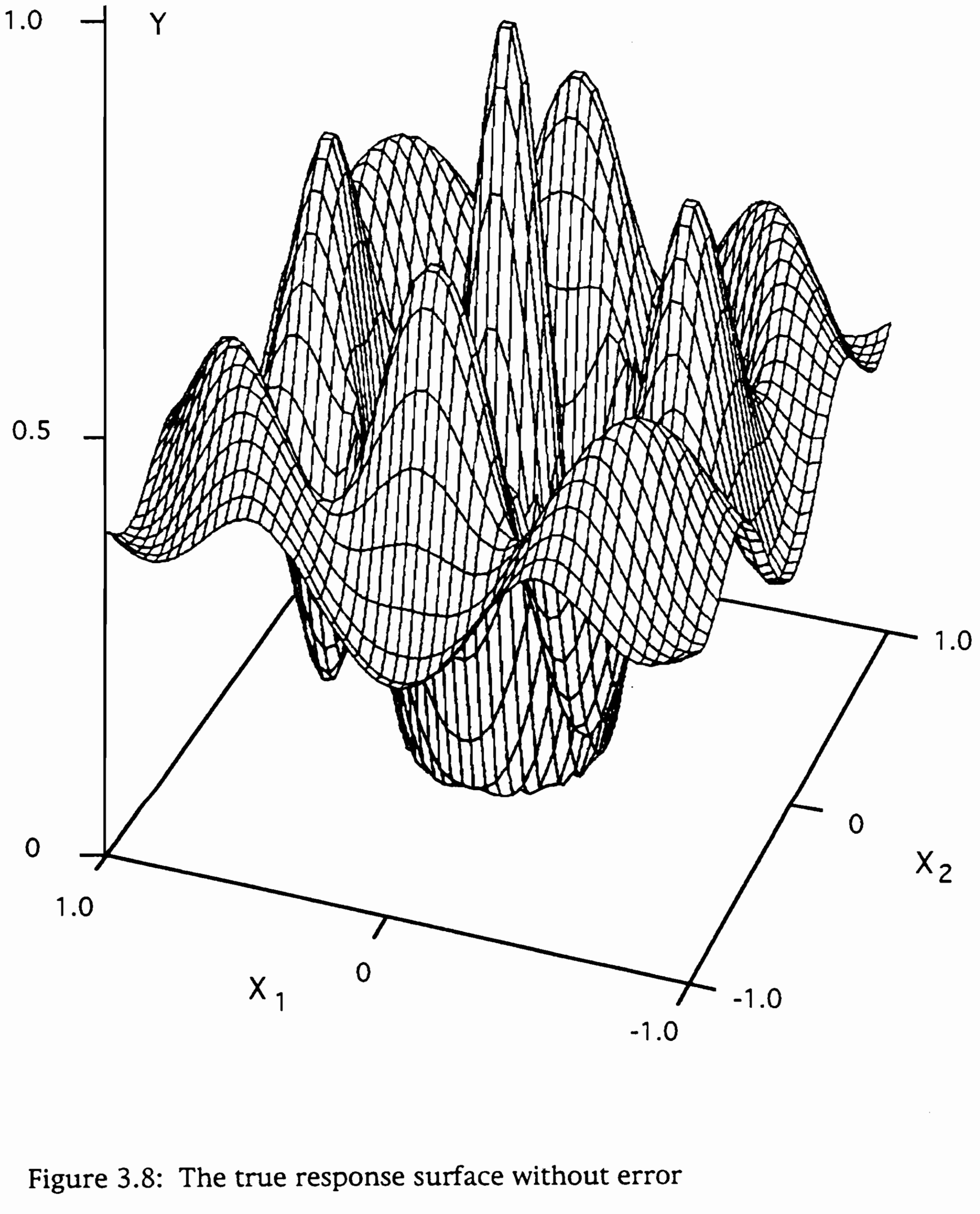


Figure 3.8: The true response surface without error

The Controller first invokes the **USER** knowledge base (figure 3.5). By way of several rules, the user is then asked to answer several questions about the model. The user responds that there are two controllable, continuous inputs x_1 and x_2, both limited to the range [-1, 1], and that the number of computer simulation runs should be limited to 150. The user also stipulates that an output value as large as possible, but at least as large as 0.85 is desired (i.e., $y > 0.85$). The User KB fires several rules to translate this information into a form needed by other modules, thereby concluding that the number of inputs is small and that the number of simulation runs is large. Since all four outputs shown in figure 3.5 have been determined, the User KB saves all facts into the KBSOS Data Base (see figure 3.3), whether obtained directly from the user or inferred by the user rules.

The Controller then chains to the **CLASSIFIER** knowledge base. The *Simulation Shotgun* component of the Classifier (see figure 3.5) uses rules and a simple FORTRAN program stored in the Methodology Base to determine the number and positioning of the shotgun simulation runs. In this case, a set of 97 runs is planned at 77 points on the surface. Of these runs, 64 are to be made at grid points, 13 are to be run at random (non-grid) points, and 20 repeat runs (i.e., multiple runs at given points) are suggested so that the variance of the true surface may be estimated. A *Shotgun* KB rule then initiates two interface programs that make the recommended simulation runs and translate results into a form usable by the *Surface Synthesis* component. A third program performs variance calculations based on the 20 repeat runs. This program indicates an estimate of surface variance of 0.003 (which is close to the true variance of ε, namely 0.05^2).

The second task of the *Simulation Shotgun* KB is to determine the input levels at which the synthesizer will be used to estimate response values, and then prepare these data for use with the neural network. As with the shotgun simulation runs, this is done by means of a rule and two programs. (Both procedures are resident in the Methodology Base.) Finally, the *Shotgun* KB places all new facts into the KBSOS Data Base.

As mentioned, we have implemented the *Surface Synthesis* module with a backpropagation neural network using an orthogonal polynomial representation of inputs. Since this example has two in-

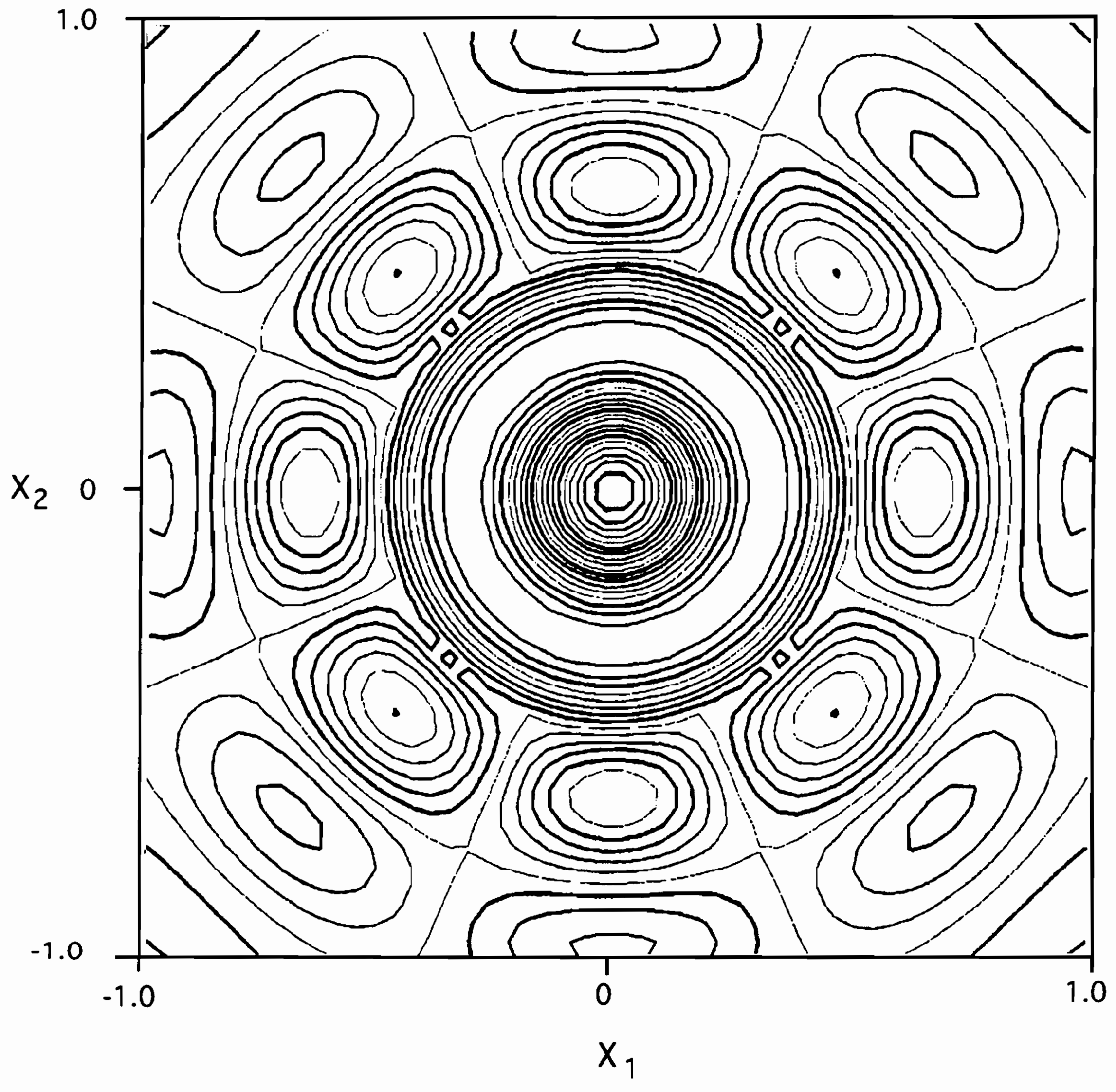


Figure 3.9: Contour plot of the true surface

put variables and one output, the neural network input layer has 23 input nodes and one output layer node (to predict the response, y). The example also uses ten middle-layer nodes. The software package NeuralWorks Professional II (Klimasauskas, 1989) is used to carry out the training of the neural network.

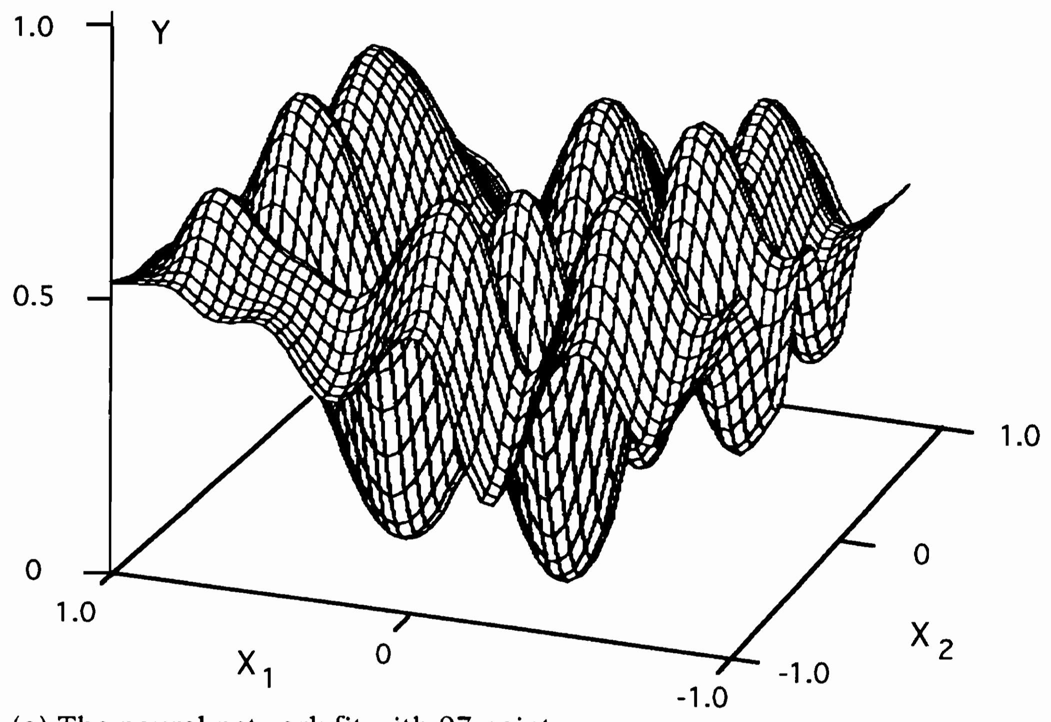
For the example at hand, after 10,000 random training presentations of the 97 data points, the estimated surface indicated in figure 3.10a is generated by the neural network. First, note the excellent fit, in spite of the highly irregular surface. Second, observe that the neural network was trained with noisy data, i.e., data with random error in it. Finally, note that the center peak is much lower in the estimated surface than in the true surface. This is due to the relatively few data points, the complexity of the surface, and the presence of error.

Now the trained network can be used in recall mode to estimate points anywhere on the surface. This is useful in carrying out the final step in classification, *Surface Characterization.* As explained above, this is accomplished by Classifier rules in conjunction with FORTRAN programs stored in the Methodology Base. The values of the four surface characteristics obtained after firing these rules and executing the programs are

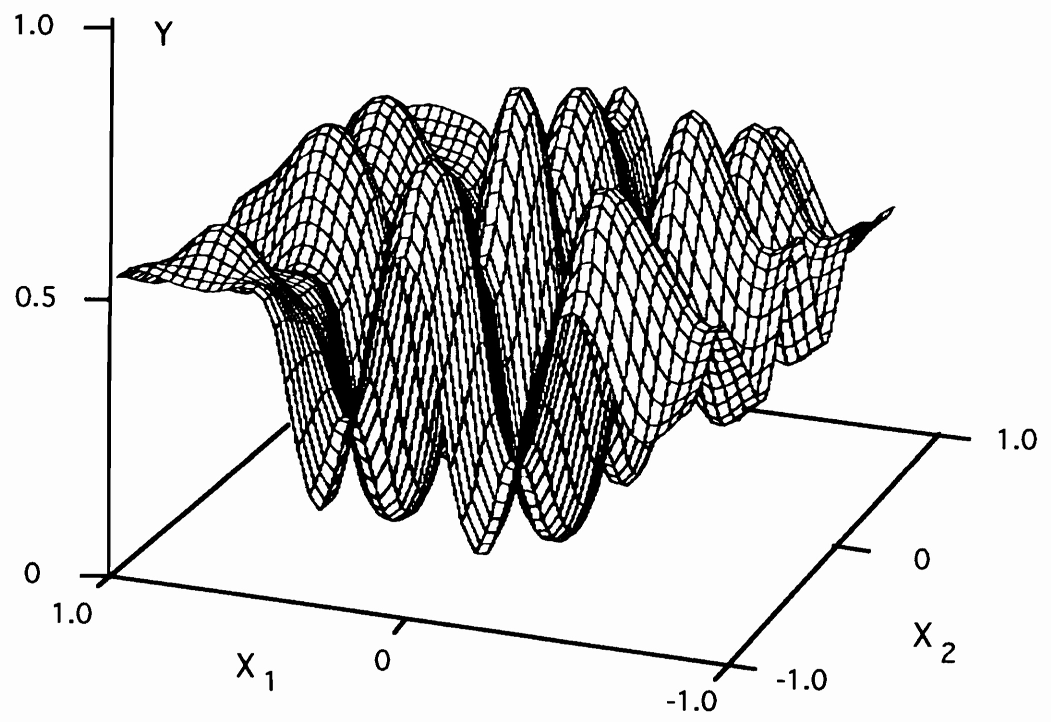
```
local optima = present
random error = small
dist_to_opt = far
factor_activity = high.
```

It is interesting to note that in determining whether local optima were present or absent, the Classifier found nine local optima. As can be seen in figure 3.8, that is precisely the actual number of significant local maxima.

The Classifier, at this point, records all its findings (including the neural network weights) in the KBSOS Data Base.



(a) The neural network fit with 97 points



(b) The neural network fit with an additional 13 points

Figure 3.10: The neural network estimates of the surface

The Controller next chains to the **STRATEGY SELECTOR** knowledge base. The Strategy Selector then accesses working memory to utilize the four parameter values determined by the Classifier plus the two other values from the User knowledge base in order to select a strategy. These six parameters are

num_factors = small
num_sim_runs = large
local_optima = present
random_error = small
dist_to_opt = far
factor_activity = high.

Consequently rule 11 shown in table 3.1 fires, and random search is chosen as the appropriate search strategy. This fact is placed in working memory as well as the KBSOS Data Base.

Since the "point" of this example is not to demonstrate the particulars of random (or any other) search method, we present the rest of the example in overview fashion. The controller next calls the **STRATEGY DETAILER**, which fires a rule and executes the REGION procedure mentioned above. Consequently, a search region of $-0.76 \leq x_1 \leq 0.95$ and $-0.85 \leq x_2 \leq 0.85$ is specified, as shown in figure 3.11a. Note that the Strategy Detailer has cut out all of the lower-valued outer perimeter, while maintaining the region encompassing all nine peaks. A random search is then conducted within the search region. A procedure in the Methodology Base concludes that an additional 13 random points should be simulated.

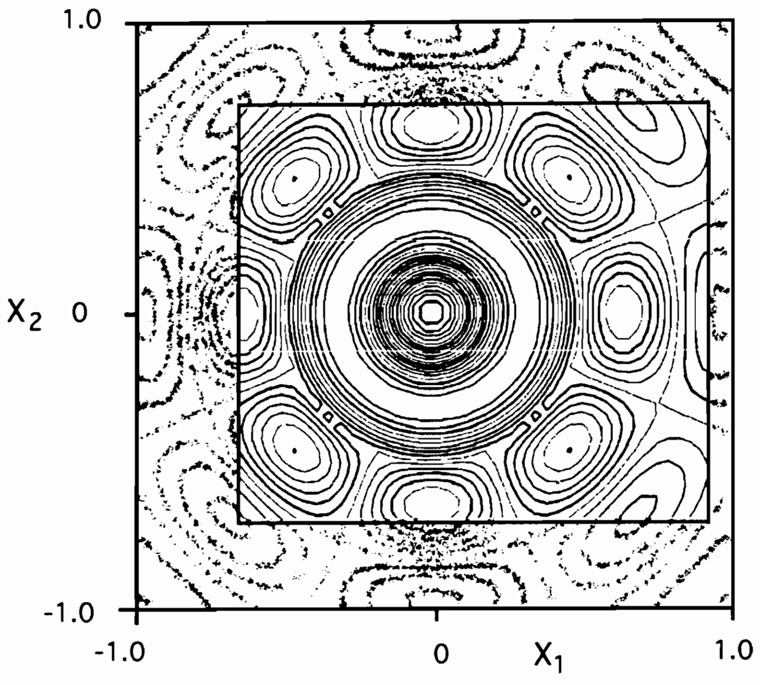
The Controller next orders the **SEARCH** to be performed as directed by the Strategy Detailer. Results from these runs are stored in the Data Base.

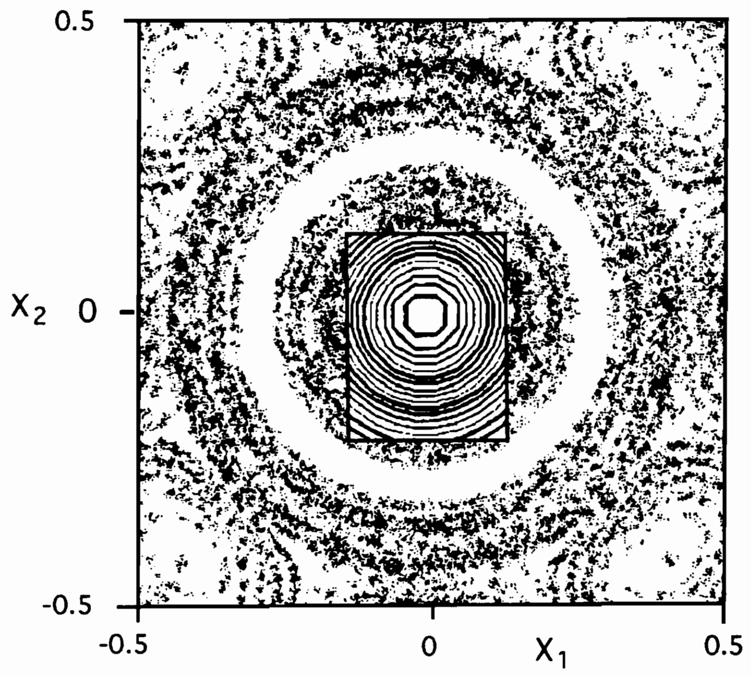
It is critical to note that what happens next with the Classifier KBSOS differs from the process for a conventional simulation optimization system. In the Classifier system advanced in this research, all (simulation) data points are now fed back through the Classifier, where another neural network surface is fit, and a new set of characterization variables specified. Therefore, it is possible that a new search strategy will be found to be preferred, which in fact is the case, as will be seen, in this example.

The Controller next passes control to the *Surface Synthesis* portion of the **CLASSIFIER**. A neural network now fits a surface through all 110 (97 + 13) data points in hand so far. The surface shown in figure 3.10b is obtained. Note that the previously-lower center peak now more accurately represents its true shape. The *Surface Characterization* rules are subsequently run, and only one surface characteristic is found to have changed in light of the 13 additional runs: local_optima now takes the value "absent." In addition, these rules conclude that the center peak is in fact significantly higher than the other peaks. This is because the neural network estimates have been combined with the actual simulation results. Therefore, only one optimum has been found during this second pass through the Classifier; it is located at $(x_1, x_2) = (0.01,-0.06)$, which is very close to the true optimum of the surface, namely $(x_1, x_2) = (0,0)$. When the **STRATEGY SELECTOR** is called, it now fires rule 32 (see table 3.1), thereby specifying that RSM I - accelerated is the appropriate search technique.

Note that a different search strategy has been proposed than was used before. This ability to switch strategies as necessary is a powerful capability of the Classifier KBSOS.

The **STRATEGY DETAILER** is called. Again our purpose here is not an exposition of RSM (the interested reader is referred to Box and Wilson (1951), Box and Draper (1987), and Myers (1971)), so we proceed in summary fashion. The Strategy Detailer specifies RSM with a full-factorial design over the region $(-0.13 \leq x_1 \leq 0.15; -0.20 \leq x_2 \leq 0.14)$. (See figure 3.11b for a contour plot of the reduced region; note the changed scale there). Seven additional runs are made (two for each factor plus three additional center points). An ANOVA (from the Methodology Base) indicates that the first-order model attempted does not provide a good fit. Therefore, four axial points and another center point are added and a second-order model is run. A global optimum is then indicated at $(x_1,x_2) = (0.005,-0.002)$, which is very close to the true optimum of (0,0). The actual height at this





(b) Region of search specified after fitting the surface with an additional 13 points

Figure 3.11: Contour plots indicating regions of fit

point (with surface error) is $y = 0.948$; without surface error it is $y = 0.999$; the true optimum at the origin is $y = 1.000$.

The Controller terminates the search because the program believes it has found the global optimum and because the user-specified goal $y > 0.85$ has been achieved.

The key feature of the example just presented is that the global optimum was found, even in the presence of many local optima, and that this was made possible by the Classifier KBSOS's ability to shift strategies intelligently as the search progressed. A single strategy pursued from the start, such as RSM, would probably have not found the true optimum unless it stumbled serendipitously onto the proper region.

Conclusions

A classification scheme has been proposed whereby a given simulation problem is characterized by the response surface it generates, and then a search technique is chosen based on that characterization. A knowledge-based system is defined that automates this "classify - search" process.

There are several limitations to the work described here, which in turn suggest many possible extensions. These include:

- complex, discrete surfaces are yet to be examined
- only six surface characteristics are considered to date
- additional search techniques can be studied
- the mapping of surfaces to techniques awaits further development, and
- additional work can be done investigating surface synthesis.

It should be noted that a framework has been developed around which these types of research issues can be investigated.

Moreover, a KBSOS has been developed which has been shown to be effective on a sample problem, and the potential advantages over single-technique methodologies have been illustrated.

Finally, we believe that this framework should now be extended to include a learner scenario, since this extension will be of such fundamental importance. This is the subject of the next chapter.

Chapter 4: Machine Learning in the Simulation-Optimization Domain

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 4.1 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.

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 annealling 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. He 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 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."

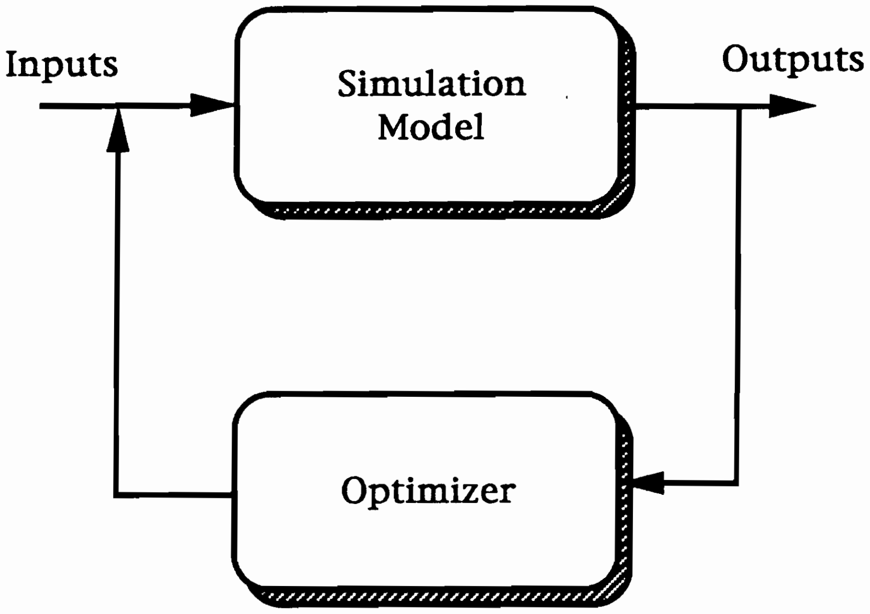


Figure 4.1: The simulation-optimization process

In a paper regarding selection of appropriate optimization technique, we (Greenwood, Rees, and Crouch, to appear) pointed out that there is both art and science in simulation optimization. We 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 we suggested consists of an inference engine, a knowledge kernel, and processing support modules (see figure 4.2). The knowledge kernel described, 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 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 and taxonomies

To provide a clearer indication of what we mean by learning in a simulation optimization context, we look briefly at the machine learning literature. In doing this, we will first look at several definitions of learning and then at two taxonomies of learning.

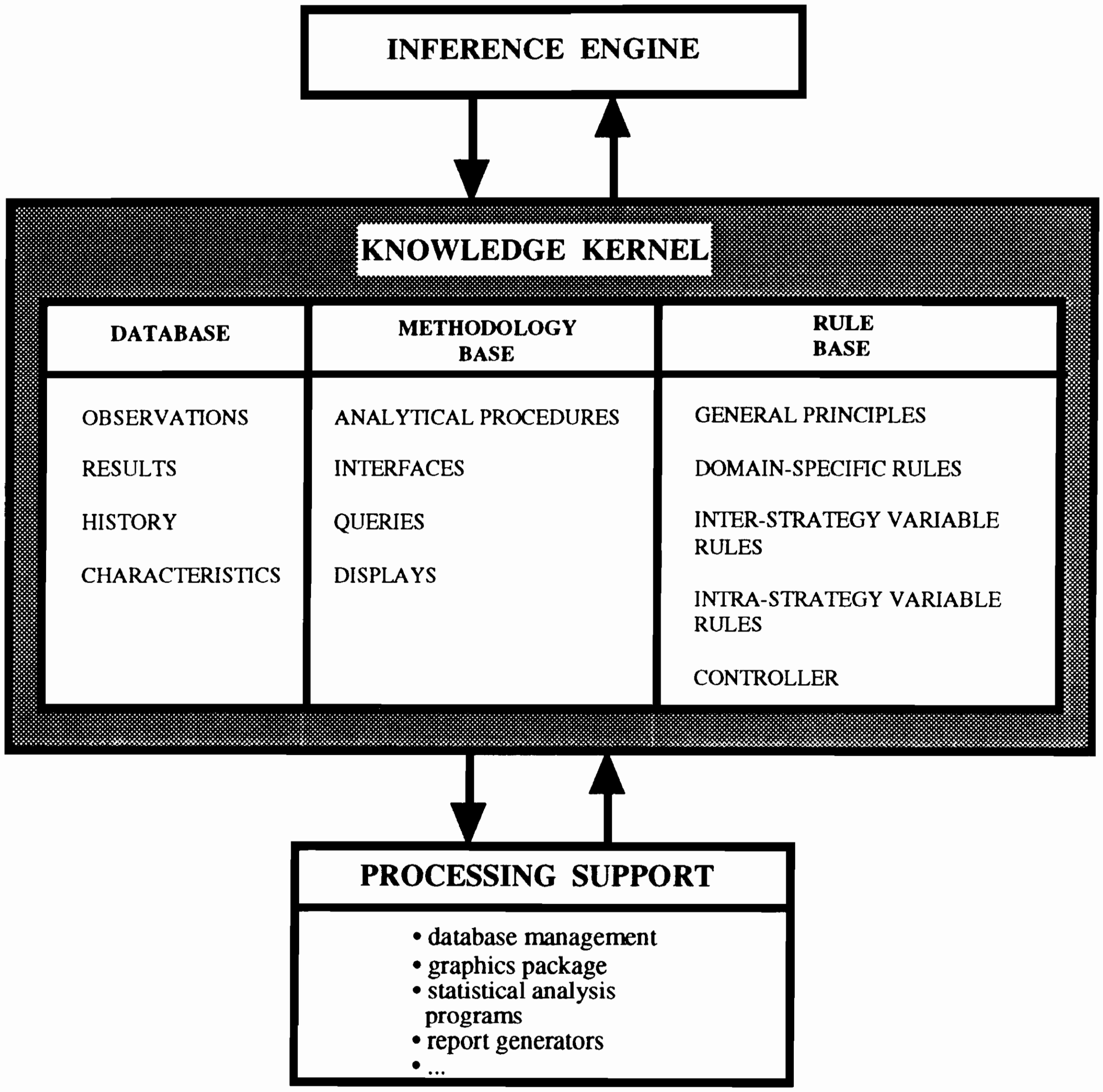


Figure 4.2: Greenwood-Rees-Crouch simulation optimization architecture

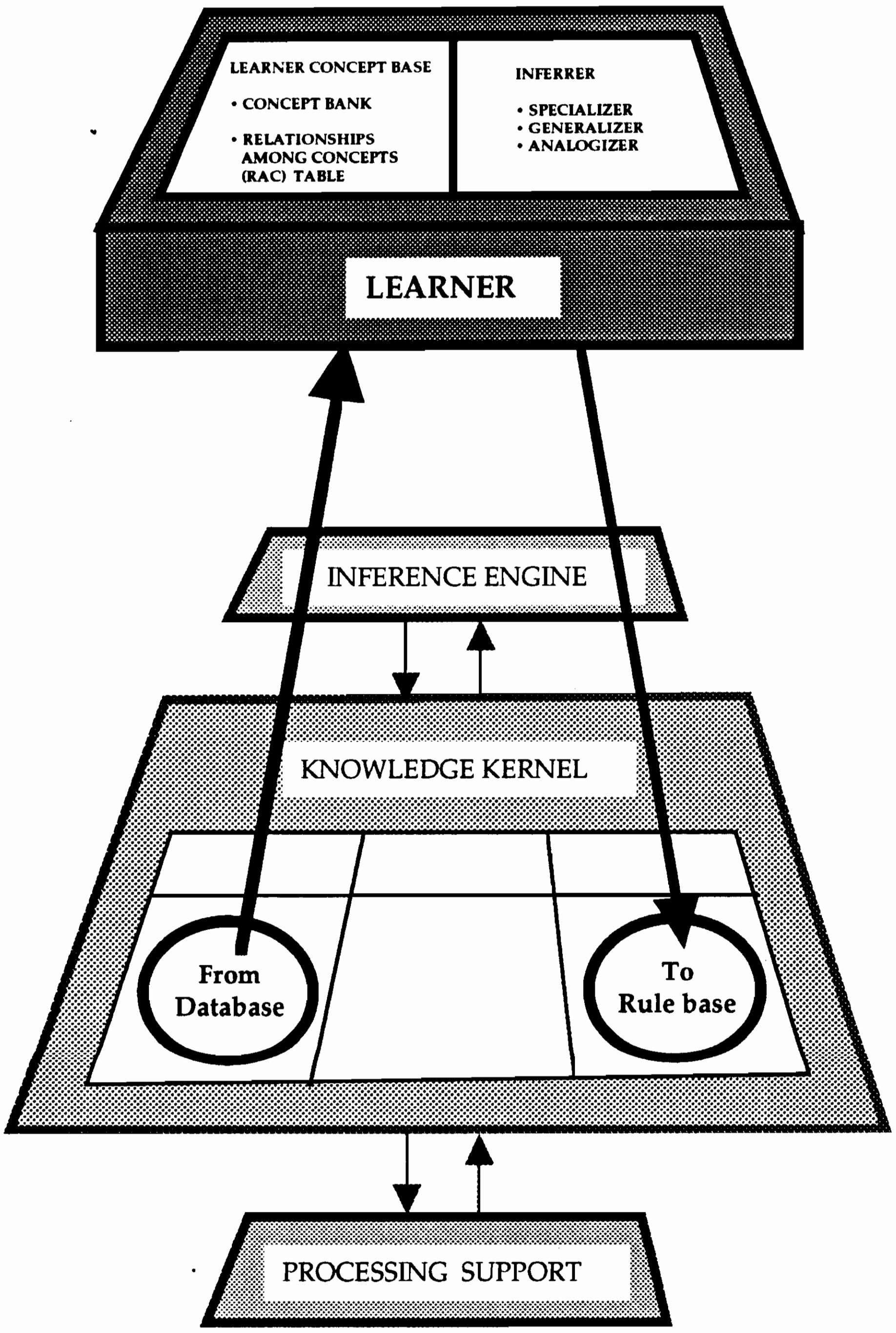


Figure 4.3: Visualization of the Learner and its environs

Various definitions of learning abound, with little apparent agreement among them. For example, Simon (1983) states: "Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more effectively the next time." Minsky (1985) in his *The Society of Mind* requires less precision by only stating that "learning is making useful changes in our minds." 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."

The Simon and Michalski definitions are closest to what we mean when we say we will let our optimizer learn. The satisficer/optimizer should be able to adapt its performance so that it improves its optimization on scenarios it has already seen. In addition, a satisficer 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.

Several taxonomies of learning exist, and it is helpful to examine two of these in order to specify later the architecture of our learner. Both of these classifications were developed by Carbonell, Michalski, and Mitchell (1983).

The first classification scheme suggested by Carbonell et al. is a taxonomy based on the learning strategy. They cite five categories, ranging from no inference required on the part of the learner, to unsupervised learning (i.e., learning without a teacher). The five categories are as follows: rote learning and direct implanting of new knowledge, learning from instruction (such as from a teacher or a textbook), learning by analogy, learning from examples, and learning from observation and discovery. Although each of these types of learning strategy could be included in a simulation optimizer to some degree or other, we believe learning by examples has the most immediate promise. In such a case, the learning source or set of examples to learn from would be the history of previous runs and results stored in the database. These runs and results could be classified and analyzed for patterns, significant factors, those parameters which seemingly are insignificant in a region, etc., as will be explained later. Simulation optimization by its very nature generates instance after instance of examples -- some "positive" examples (demonstrating the concept to be acquired) and others "negative" examples (which are useful in preventing overgeneralization).

The second scheme we use from Carbonell et al. is a classification based on the type of knowledge acquired. The concepts we use from this taxonomy include rule modification or creation, specialization, parameter modification, and 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. Rule and parameter modification are described below.

Using the Simon and Michalski definitions and the two Carbonell et al. taxonomies, we may now state what we mean when we say that a knowledge-based simulation optimizer will "learn." Learning as used here is the adaptation or modification of representations of knowledge stored in an optimizer's knowledge base in order to improve future simulation optimizations on examples or scenarios close to those already seen. This adaptation or modification will occur through specialization, generalization, parameter modification, and rule modification.

A search of the literature indicates a total void to date in the application of machine learning concepts to improve simulation optimization. This research begins to fill that void. The contribution of our research is two-fold. First, a general learner architecture for simulation optimization is developed. This architecture is specific enough to show what there is to be learned in a knowledge-based simulation system and to illustrate how it can be done. Second, some answers to what we believe are key implementation considerations are provided and discussed by way of example.

The remainder of this chapter is organized as follows. First, the general learner architecture is presented. This is done by way of a generic learning flow diagram. Then what there is to learn in a knowledge-based simulation optimization system (KBSOS) is pointed out. Next, implementation considerations are examined. Finally, an example to illustrate both the architecture and implementation issues is presented.

General Learner Architecture

Overview

The learner architecture developed here includes four types of learning: specialization, rule modification, parameter modification, and generalization. Specialization and generalization have been described previously as the addition to and deletion from rule antecedents. 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 annealling is preferred. Thus, the modified rule would have the consequent "THEN strategy = simulated annealling." Additional types of learning can be added to the Learner later if desired as plug-in modules; however, at this point, it is our judgment that these four types of learning should suffice.

Each of the four learning types to be included in the 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, therefore, is to separate the "art" and "science" in the learner, just as we did in the KBSOS. This is indicated in figure 4.4.

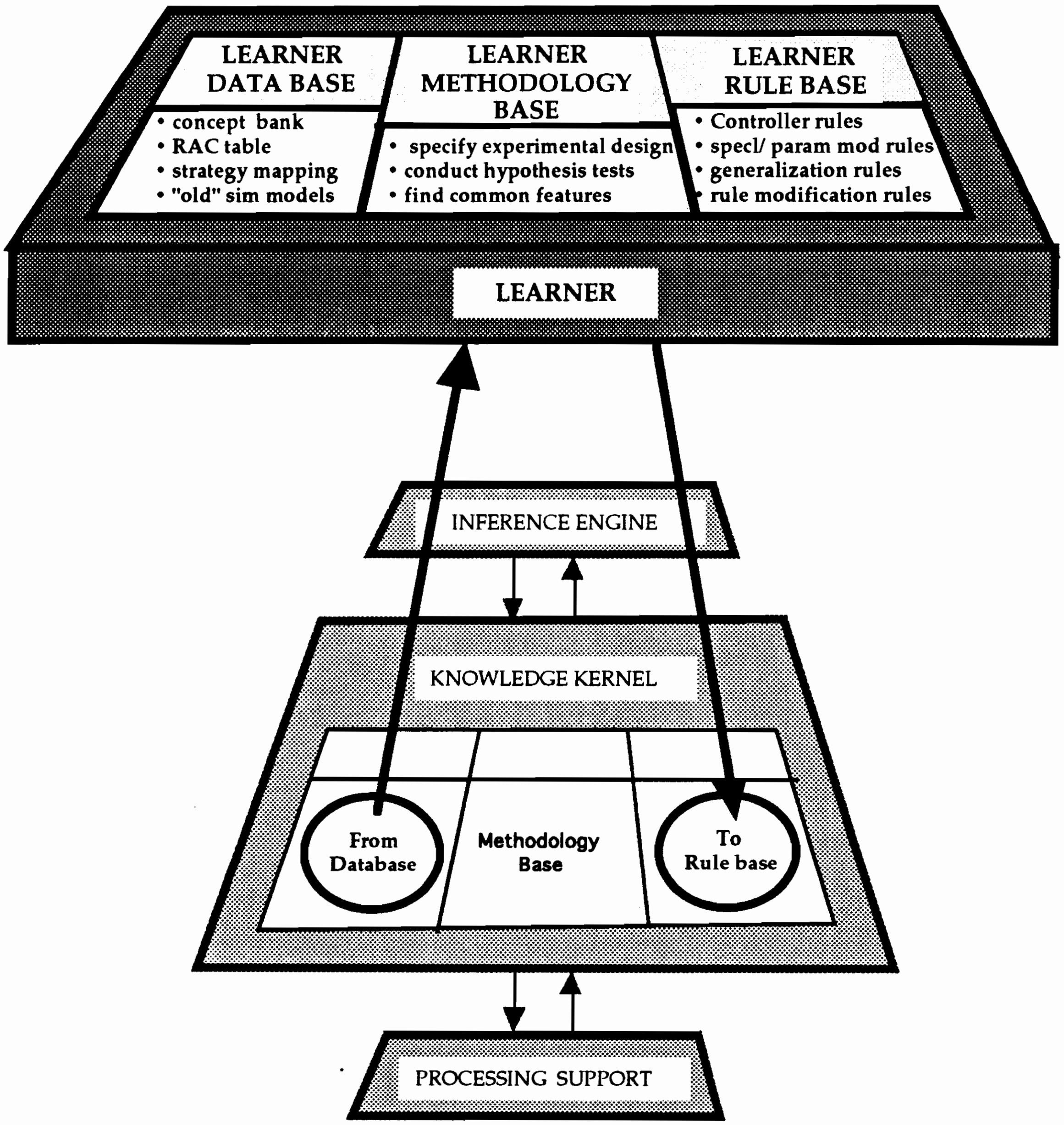
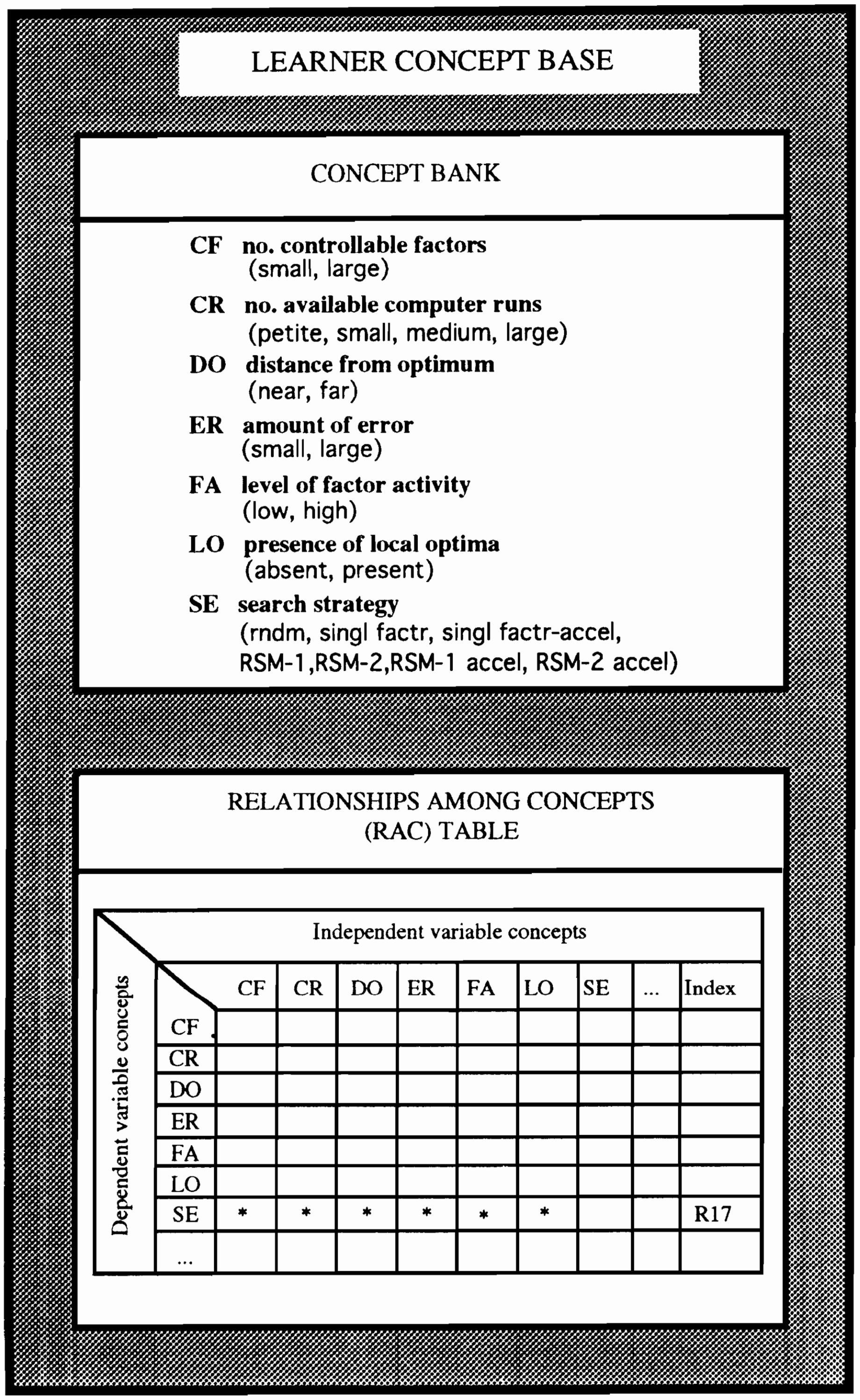


Figure 4.4: The Learner architecture

Figure 4.4 shows the learner sitting above the KBSOS and deriving input from the KBSOS database; changes are passed back to the KBSOS rule base, as in figure 4.3. But figure 4.4 shows explicitly 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 4.4 shows some of the functions to be carried out by each of the three modules.

A knowledge-based simulation optimization system contains many concepts that may be stored in a variety of 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. We first postulated a mechanism for doing this in Greenwood et al., where we suggested a concept bank and a Relationships Among Concepts (RAC) table. We repeat an expanded portion of this in figure 4.5, where a concept bank is shown that maintains a list of concepts used in rules, etc., together with the levels at which those concepts appear. The RAC table stores which concepts are used in which rules. As indicated in figure 4.4, 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 the chapter. An additional item included in the Learner Data Base is a collection of "old" simulation programs. That is, whenever a simulation program is run and its results are 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 may not be 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 the Learner Data Base may share or coincide or differ from the knowledge kernel data base, and that what has been described above is not necessarily a physical representation.

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



LEARNER CONCEPT BASE

CONCEPT BANK

CF	**no. controllable factors** (small, large)
CR	**no. available computer runs** (petite, small, medium, large)
DO	**distance from optimum** (near, far)
ER	**amount of error** (small, large)
FA	**level of factor activity** (low, high)
LO	**presence of local optima** (absent, present)
SE	**search strategy** (rndm, singl factr, singl factr-accel, RSM-1,RSM-2,RSM-1 accel, RSM-2 accel)

RELATIONSHIPS AMONG CONCEPTS (RAC) TABLE

Independent variable concepts

Dependent variable concepts

	CF	CR	DO	ER	FA	LO	SE	...	Index
CF									
CR									
DO									
ER									
FA									
LO									
SE	*	*	*	*	*	*			R17
...									

Figure 4.5: The learner in the context of the Smith-data example.

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. Although it is recognized that these submodules are complex, they can be implemented using ideas well established in the literature, so we do not provide additional elaboration here. A third submodule in the learner methodology base will be explained later and deals with searching for common features or concepts for a given set of rules.

The 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. These different rule sets will be discussed further shortly.

Having specified a general architecture for a learner, we now discuss the process to be followed in learning. This is done in terms of a flow diagram in the next section. A detailed example illustrating the concepts developed here and in the rest of the research is included at the end of the chapter.

General learning flow diagram

In order to develop the process of learning in a KBSOS, we borrow with minor modifications a learning paradigm from the realm of case-based reasoning (CBR) developed by Slade (1991). As our research at this point is not proposing any adaptation of any CBR concepts per se at all, we will not discuss Slade's CBR context. Rather, we will only present the outline of his learning paradigm in the context that we will use it.

Figure 4.6 indicates the learning process we have adapted from Slade. The shaded boxes indicate the major operations in the process needed for all four learning types. (The only exception is that we do not need **Repair** 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, (as will be explained later) 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 4.6 indicates.

As mentioned, the general learning process for all four kinds of learning is the same and follows the flow shown in figure 4.6. The particular process used to "determine learning type," the top box in that figure, will be discussed after additional concepts have been developed. In addition, a detailed example illustrating specialization and rule modification will be presented later.

Referring to figure 4.3 again, we have now presented in a general way an architecture and a process flow for the Learner, the top box in that figure. We now turn our attention to a lower box in figure 4.3, the knowledge kernel, to develop further the context of how and where learning should occur in a KBSOS. After that, we will return to the learner for additional details.

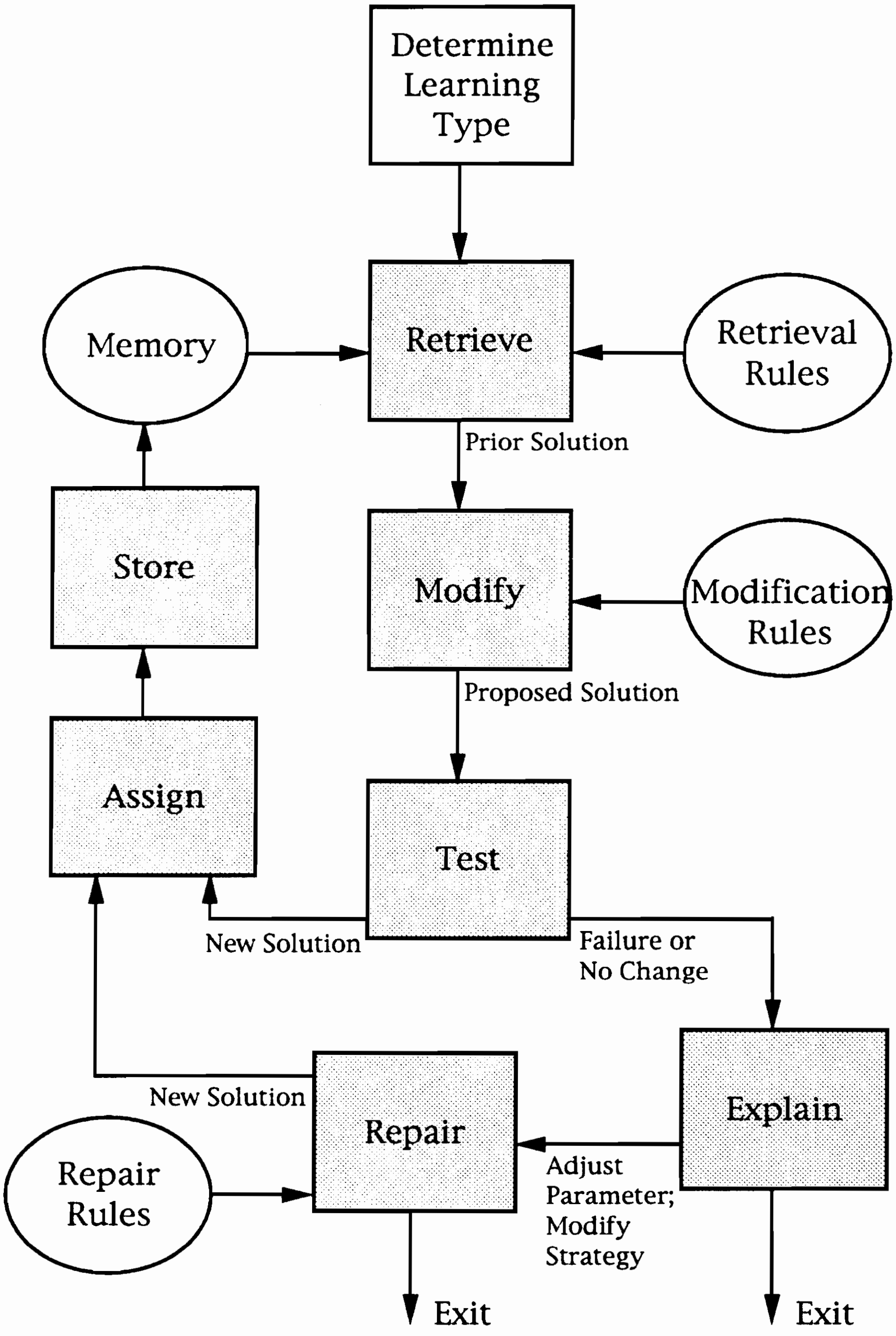


Figure 4.6: The learning process

What There is to Learn in a KBSOS

As mentioned, the idea of a knowledge-based simulation optimization procedure was first advanced in Greenwood et al. In chapter 3, we fleshed out these ideas and built a prototype KBSOS using a popular knowledge-based shell (VP-Expert (1989)), a neural network package (NeuralWorks Professional II (Klimasauskas, 1989)), and various third-generation routines. We called this a "Classifier KBSOS," because its simulation output surfaces are ***classified*** according to the search strategy most likely to render success. We present now a quick overview of the Classifier KBSOS process in order that we might show where in that process learning is to occur and to demonstrate how this is to be done.

In the Classifier KBSOS, input sufficient to define the problem is obtained from the **User** (see figure 4.7). This input is then fed to the **Classifier**, where three steps occur. First, the "shotgun" suggests 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.) 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. We have used the same explanatory variables he used in his study to classify our surfaces with the **Classifier**.

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

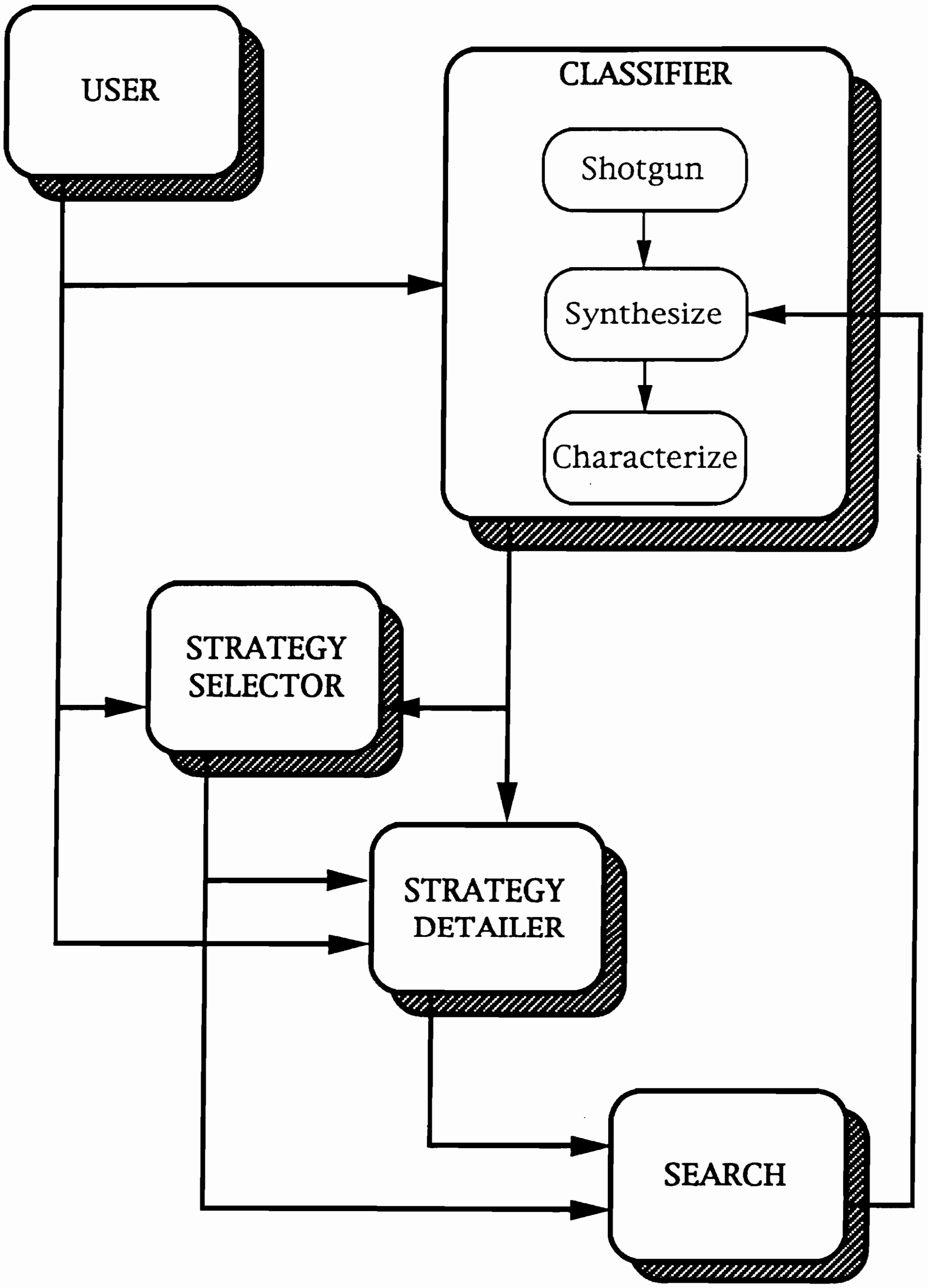


Figure 4.7: An overview of the Classifier KBSOS

strategy has been chosen, the **Strategy Detailer** (another set of rules) is fired, and implementation particulars are set whereby the **Search** may be conducted.

It should be clearly stated what is *not* meant when one suggests that a KBSOS will learn. The learner is not expected automatically to deduce or infer a new 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, some examples of each of the four kinds of learning are briefly listed:

> *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 δ 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 re-specify "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 that we have taken personally 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.

Implementation Considerations

We now return to the top box of figures 4.3 and 4.4, the Learner, to provide more insight into the learning process. In particular, we discuss timing, i.e., how often should learning occur, and consider what cases on which to learn. Finally, we give an overview of a learning session. This includes specification of the Controller, that portion of the learner rule base (see figure 4.4) that selects the type of learning to attempt.

Timing

The approach to simulation optimization learning advocated here does not permit the changing of any rules during a simulation study. Stated differently, learning is only permitted between simulation studies. This is indicated in figure 4.8, where the horizontal axis is time. In that figure each small arrow indicates a complete simulation study with its myriad of computer runs and searches for optimum or satisficing conditions. Learning sessions are indicated in the figure by long arrows. Note that these sessions occur rather infrequently and do not coincide with any simulation studies.

Note also in figure 4.8 the letters "G," "B," and "U." These letters relate to the issue of which cases to learn on. In general, one may choose to learn from all historical cases, only the most recent cases (e.g., since the last learning session), or only from "interesting" historical cases, etc. After the da-

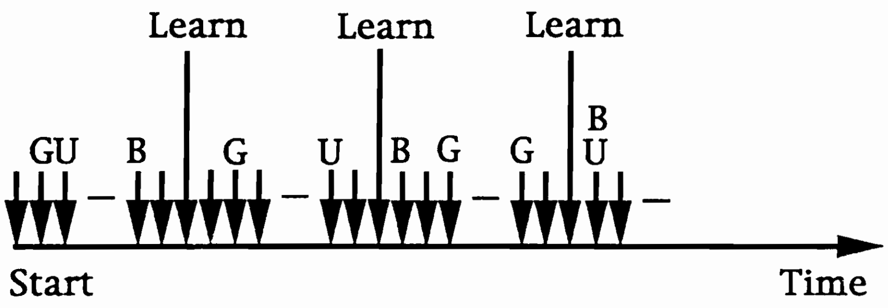


Figure 4.8: Marking simulation optimization cases for learning

tabase of previous studies has grown, it may become prohibitive to examine all historical cases, so we examine here one way of defining and using "interesting" history. We define "interesting" in terms of both efficiency and effectiveness in finding the optimal (or near-optimal) response for each given simulation study. Our chosen measure of efficiency is the total number of simulation runs used to find the optimal response. Hence a case that used relatively few simulation runs would be marked ***Good*** (or "G"), whereas a case that used relatively many runs would be marked ***Bad*** (or "B"). This is not to say that every case that uses many runs is bad; a very convoluted surface may legitimately require a lot of runs. The purpose in marking cases as "interesting" is to identify cases on average which have the potential of leading the Learner to information that has the most promise for improving future simulation optimizations. Our chosen measure of effectiveness is the degree of confidence we have that the solution we found is the "best." In this case, we label as "interesting" those cases in which we have less confidence in our answer; we determine our confidence by observing the variance of the surface and whether multiple optima are present. If there is high variance or if multiple optima exist, we label this case ***Ugly*** (or "U").

The learning session

The final major item to be specified in the Learner architecture and flow is how to determine which type of learning to attempt when. Stated differently, it remains to define the control of the learning system. In the context of figure 4.4, the controller rules must be explained; this is the same function as the top box in figure 4.6, "Determine Learning Type." We will define the Controller for examining only "interesting" cases as outlined above; other possibilities such as examining all cases or reviewing only recent history are easier to specify, and are omitted here.

The Controller flowchart for examining interesting history is shown in figure 4.9. Three basic questions are asked sequentially in that figure: of the marked (as interesting) historical cases, is there a most frequently occurring search strategy; of all the parameters set in the knowledge kernel, is there an unacceptably low confidence in any one; and, is there a need to consolidate rules, or to "houseclean," as it is stated in figure 4.9.

The precise implementation of the Controller depends on the definition of "interesting" history. In this discussion, we will use our translation of "interesting" as either ***Good***, ***Bad***, or ***Ugly***, but the basic Controller flow remains essentially the same regardless of the particulars of the choice. In essence, the Learner needs to know if there is an especially successful or a particularly unsuccessful set of cases tied to a particular search strategy.

The first question asked by the Controller is whether there is a most frequently occurring search strategy within each subgrouping of interesting history. If so, and the interesting cases are marked ***Bad***, or ***Ugly***, then this search strategy is a candidate for rule modification. If conversely, the interesting cases have been marked ***Good***, then the question becomes, why didn't all cases work well with this strategy. I.e., the question is, why did just the interesting cases work well with this strategy, and not the uninteresting (or unmarked) cases. Is there some as yet unidentified concept that distinguishes the domain of interesting cases from that of the uninteresting history? But answering this question and finding the antecedent concept that distinguishes the interesting and uninteresting histories is just specialization. Thus, the Controller asks its first question about most frequently occurring search strategy, and if one exists for any or all of the ***Good***, ***Bad***, or ***Ugly*** cases, the Specialization and Rule Modification module is called.

Figure 4.10 is the flow process for Specialization and Rule Modification, and is similar in structure to figure 4.6, the flow diagram for the general learning process. Because of the similarity to figure 4.6, which has already been explained, discussion of figure 4.10 will be delayed until the next section, in which a detailed example is presented.

The second question asked by the Controller in figure 4.9 is whether there is a parameter in the knowledge kernel that should be examined. The answer to this question may be based upon a ranking of parameters by the developer, for example, from most uncertain to most certain; or, it

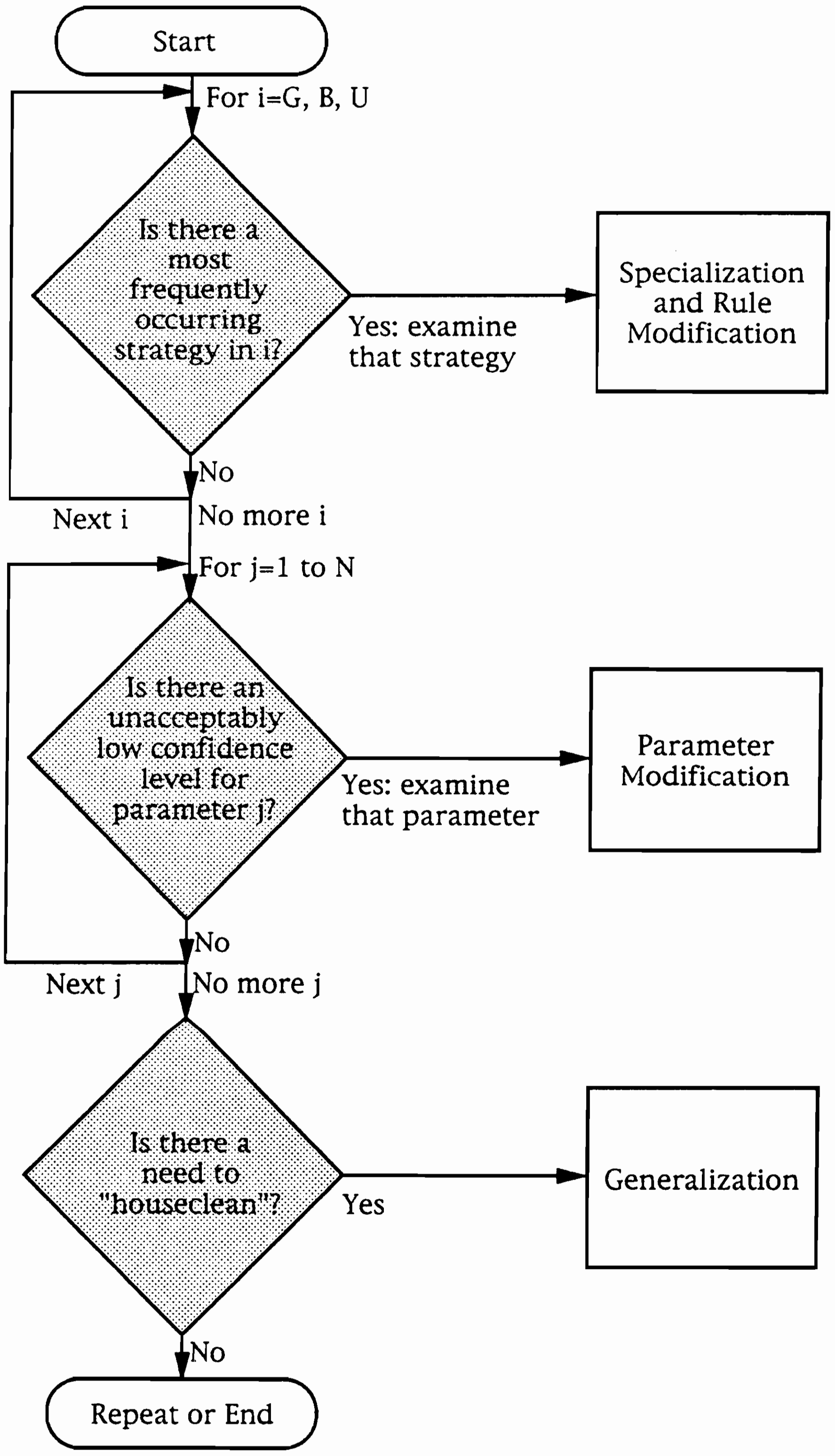

Start
For i=G, B, U
Is there a most frequently occurring strategy in i?
Yes: examine that strategy
Specialization and Rule Modification
No
Next i
No more i
For j=1 to N
Is there an unacceptably low confidence level for parameter j?
Yes: examine that parameter
Parameter Modification
No
Next j
No more j
Is there a need to "houseclean"?
Yes
Generalization
No
Repeat or End

Figure 4.9: Controller flowchart

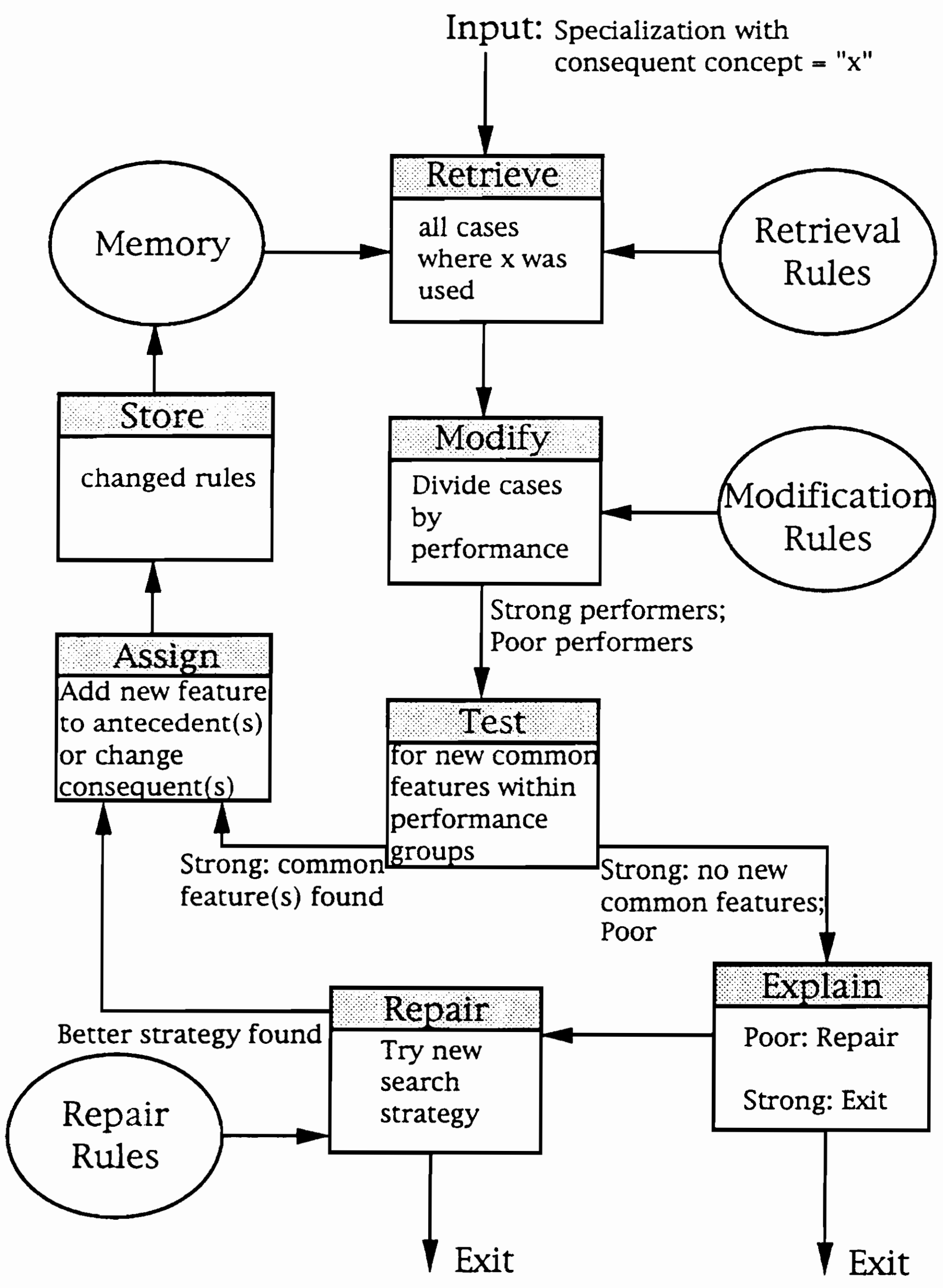

Input: Specialization with consequent concept = "x"
Retrieve
all cases where x was used
Memory
Retrieval Rules
Store
changed rules
Modify
Divide cases by performance
Modification Rules
Strong performers; Poor performers
Assign
Add new feature to antecedent(s) or change consequent(s)
Test
for new common features within performance groups
Strong: common feature(s) found
Strong: no new common features; Poor
Explain
Poor: Repair
Strong: Exit
Repair
Try new search strategy
Better strategy found
Repair Rules
Exit
Exit

Figure 4.10: The specialization and rule modification process

may be based on a simple list of parameters about which the developer is unsure. Regardless of implementation, if any parameter is judged as worrisome, then the "Parameter Modification" module is called. This module is specified in figure 4.11. As that figure shows, the rule with that parameter is retrieved and then temporarily modified. Relevant known surfaces are also retrieved. The learner then tests the old and new values by carrying out simulations as needed. The learner methodology base's experimental-design submodule and hypothesis-testing submodule would be called if required. If results show improvement, the parameter is changed to its new value and stored back in the knowledge kernel. If not, repair is attempted. If unsuccessful, the parameter modification process is exited.

The final question asked by the Controller of figure 4.9 is whether there is a need to consolidate rules. Our approach for answering this question is to reply yes whenever 25 or so rules have been modified by the learner. In this case, potential for generalization of rules exists, and the "Generalization" module is called. This module basically determines if any antecedents of existing rules may be dropped or if rules may be combined to be simplified. The generalization process is indicated in figure 4.12, but is not discussed here as we have already explained this in Greenwood, et al. (to appear).

With the Learner Controller now specified, it remains to demonstrate how the Learner works as a whole. We now show this by means of a detailed example.

Example

In this example we illustrate how the Learner we have described above can be used to change rules in the knowledge kernel of the KBSOS. In particular, we are going to demonstrate two of the four learning types in our Learner: specialization and rule modification. We will also demonstrate how

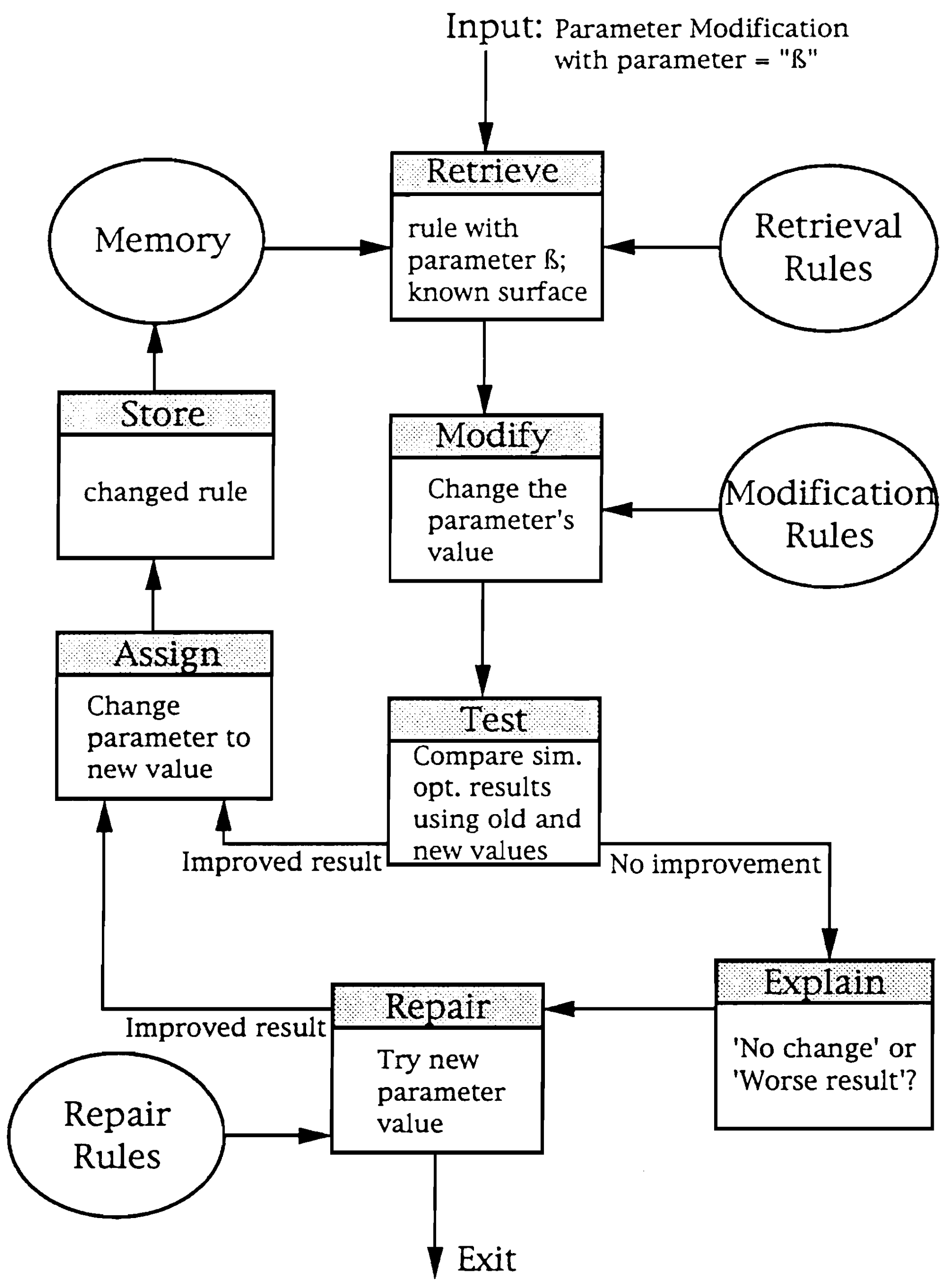

Input: Parameter Modification with parameter = "ß"
Retrieve
rule with parameter ß; known surface
Memory
Retrieval Rules
Store
changed rule
Modify
Change the parameter's value
Modification Rules
Assign
Change parameter to new value
Test
Compare sim. opt. results using old and new values
Improved result
No improvement
Explain
'No change' or 'Worse result'?
Repair
Try new parameter value
Improved result
Repair Rules
Exit

Figure 4.11: The parameter modification process

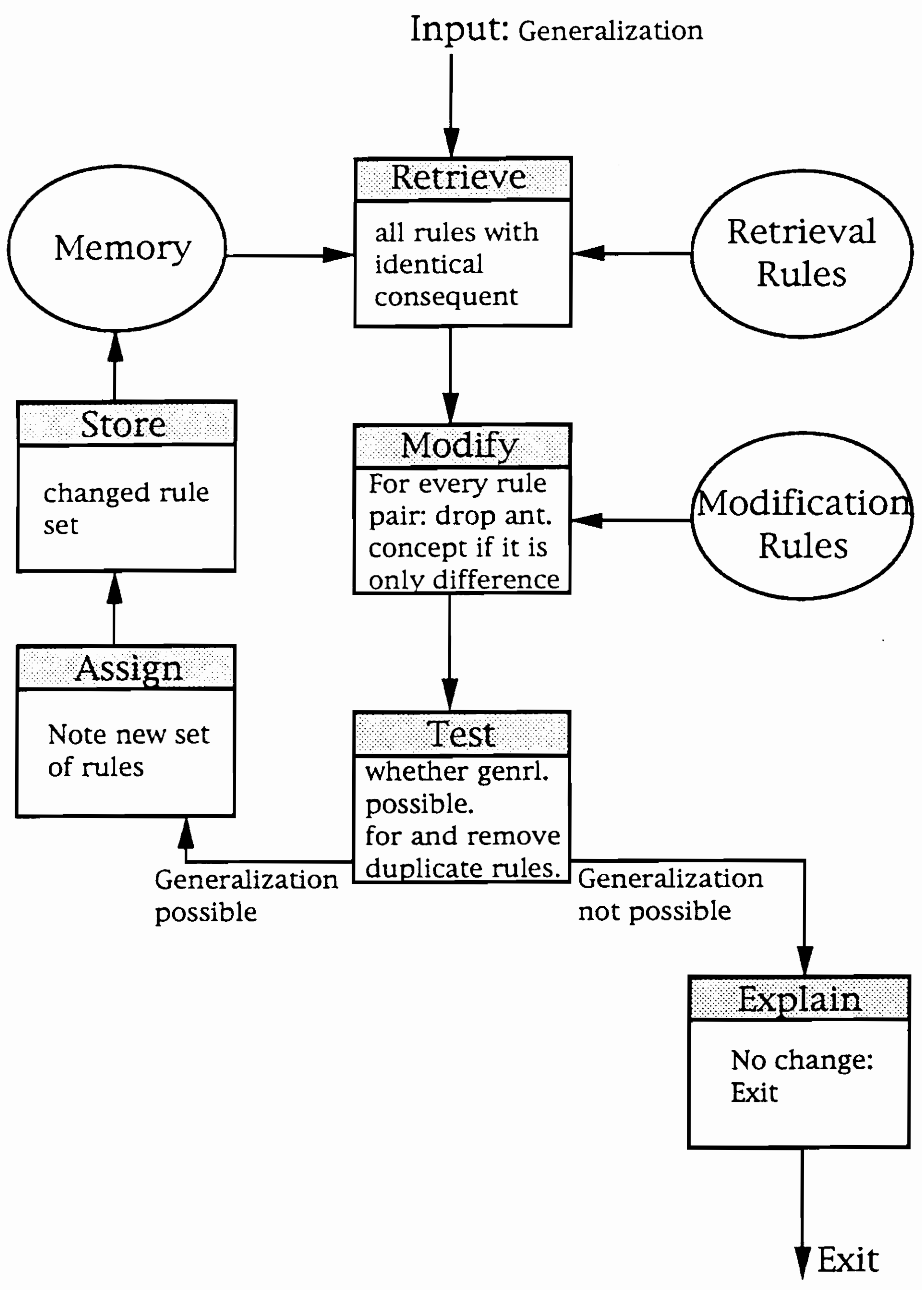

Input: Generalization
Retrieve
all rules with identical consequent
Memory
Retrieval Rules
Modify
For every rule pair: drop ant. concept if it is only difference
Modification Rules
Store
changed rule set
Assign
Note new set of rules
Test
whether genrl. possible. for and remove duplicate rules.
Generalization possible
Generalization not possible
Explain
No change: Exit
Exit

Figure 4.12: The generalization process

these steps can be implemented in an expert-system shell. (We will illustrate with VP-Expert (1989), a widely used shell.)

With reference to the Learner architecture of figure 4.4, we will begin our example with the Controller rules in the Learner Rule Base. The Controller will invoke the Specialization/Rule Modification rules in that rule base, which in turn will retrieve data from the data base in the KBSOS knowledge kernel, process it, and then change rules in the knowledge kernel rule base.

An overview of the process flow to be followed in the example is as follows: first, the Controller of figure 4.9 will be invoked. This will then lead to the Specialization and Rule Modification module in that figure being called, which is shown in detail in figure 4.10. The first step, as shown in figure 4.10, is to **Retrieve** data from the knowledge kernel; the next steps are to **Modify** and **Test** cases, and finally to **Store** changed rules back in the knowledge kernel.

Controller: Most Frequently Occurring Strategy

In the Controller of figure 4.9, we begin at the **Start** block. Activity then passes to the first Controller question, is there a most frequently-occurring search strategy specified in the interesting cases marked ***Good.*** We assume for the purposes of this example that 1000 interesting cases have been marked since the last learning session, and that 200 of these cases were marked ***Good.*** These interesting, ***Good*** cases are shown in figure 13a. They may be extracted from the 1000 total cases in VP-Expert by simply using the database "GET" command. Now among the 200 ***Good*** cases of figure 4.13a, the question is asked whether there is a predominant (search) strategy among these 200 that led to this good performance. Assume that the strategy "RSM-1" is most frequently occurring, and that 100 such records were found. Recall that the purpose of this step is to see whether there are any candidates for learning improvement, and that no learning or testing has been conducted thus far. Only the search strategy RSM-1 has been identified as a candidate for learning.

No.	Record No.	Mark	Strategy
1	1	G	RSM-1
2	8	G	RSM-2
3	21	G	Random
.	.	.	.
.	.	.	.
.	.	.	.
200	968	G	RSM-1

(a) All cases marked "Good"

No.	Record No.	Mark	Strategy
1	1	G	RSM-1
2	32	-	RSM-1
3	53	U	RSM-1
.	.	.	.
.	.	.	.
.	.	.	.
300	987	B	RSM-1

(b) All cases with "x = RSM-1"

					Antecedent Concept					
No.	Record No.	Mark	Strategy	Perf.	C1	C2	C3	C4	C5	C6
1	1	G	RSM-1	S	L1	L2	L2	L3	L2	L2
2	71	G	RSM-1	S	L1	L2	L1	L2	L2	L3
3	126	G	RSM-1	S	L2	L2	L1	L2	L2	L1
.	.	.							.	
.	.	.							.	
.	.	.							.	
20	968	G	RSM-1	S	L2	L1	L2	L1	L2	L1

(c) Antecedent concepts of "Strong Performers"

Figure 4.13: Specialization Example

Specialization: Input

The strategy RSM-1 having been identified, control now passes to the "Specialization and Rule Modification" box of figure 4.9, which is just figure 4.10. The "Input" to this figure is specialization with consequent concept = RSM-1.

Specialization: Retrieve

The **Retrieve** box in figure 4.10 then retrieves all cases in the database where x = RSM-1 was used, not just those marked as interesting and ***Good.*** In VP-Expert, this again is accomplished with another "GET" statement (e.g., GET RSM-1 = strategy, filename, ALL). Assume that as a result of the GET, a total of 300 cases is found: the 100 found above (marked ***Good***), plus 200 more. These 300 cases are shown in figure 4.13b. Note in that figure that of the 300 cases, some have no mark at all, whereas others do. The one element in common is that all 300 cases used the RSM-1 strategy.

Specialization: Modify

The **Modify** box of figure 4.10 is incurred next. This box uses modification rules to divide the 300 cases into ***strong*** and ***poor*** performers by calculating performance. Note that the issue of assessing performance is complex, in general. We believe this to be a profitable topic for future research and note that learning in general cannot occur when performance is not evaluated. For now, we use two simple modification rules to evaluate performance:

```
RULE Modification_1
IF marked = G  AND
  marked < > U
THEN performance = strong;
```

```
RULE Modification_2
IF marked = B
THEN performance = poor;
```

We assume the results of applying these two rules to the 300 cases of figure 4.13b leads to 20 ***strong*** performers being identified as well as 40 ***poor*** ones. Once these two different performance categories are determined, then VP-Expert's GET statement may be used again to group all the ***strong*** performers, for example, together, as shown in the left portion of figure 4.13c.

Specialization: Testing for new common features

Recall from the earlier discussion that the Learner's Concept Bank keeps track of all antecedent concepts used in the Strategy Selector rulebase. Moreover, it also records all levels used for each concept (see figure 4.5). In our Strategy Selector, we have used just six antecedent concepts following the lead of Smith (1973). For purposes of illustration, we therefore assume there are only six concepts listed in the concept bank together with their levels. This obviously may be easily modified.

Now extracting data from the concept bank for the ***strong*** performers identified in the **Modify** step above enables the full table of figure 4.13c to be generated. The table now contains all the information necessary to conduct the test. For example, from the third item in that table, it is seen that record no. 126 in the database was marked ***Good,*** used the RSM-1 strategy, is considered a ***strong*** performer, and has concepts 3 and 6 at level 1 and all others at level 2.

Recall that figure 4.13c is a table of historical cases taken from the knowledge kernel database; it is not a listing of rules. The test listed for specialization in figure 4.10 is to see whether new common features can be found within performance groups. Stated differently, the test mentioned in figure 4.10 is to see whether all ***strong*** performers, for example, have a concept at the same level. If so, then perhaps that concept at that level should be added to the rule base as an antecedent, because the presence of that concept at that level may well explain why all these cases performed ***strongly*** with the RSM-1 search strategy. In the example, this is easily implemented. The program must determine if any concept is only present in the cases at a single level; i.e., the learner must see if any concept column has only one level. In figure 4.13c, concept C5 is at level L2 across all ***strong*** performers; therefore, the program finds the common feature, namely concept C5 at level L2. Note that the code to find common features may be easily implemented in a procedural language and stored in the Learner Methodology Base for use as needed.

Concept C5 having been chosen as a common feature, it remains to see if this common feature is a ***new*** common feature. That is, it remains to see if there are any rules that can be altered to include this concept as an antecedent, or if all applicable rules already include this concept. This is implemented in a three-step procedure, which also is stored in the Methodology Base. First, all rules in the Strategy Selector with consequent "strategy = RSM-1" are extracted. Then, for each rule, it is seen if any case specified in figure 4.13c would have fired that rule. If this rule could have been fired by any of the cases in the figure, then we say that the rule is "chosen." Finally, once all rules are tested to see which are chosen, then chosen rules are checked to see if any lacks the antecedent "C5 = L2." If so, then the **Test** passes and control is passed to the next **(Assign)** step. If no such rules are found, then no new common features have been found, and the specialization module is exited for the ***strong*** performers' case.

Specialization: Assign

As mentioned, "chosen" rules have been identified for which specialization is appropriate by adding the new antecedent "C5 = L2." In this **Assign** step, the antecedent is added to all chosen rules.

It is also necessary in this step to add a new rule with the new antecedent "C5 < > L2," but with the same consequent (search strategy = RSM-1). This prohibits cases from "falling through the cracks" and ensures that the rules cover all the cases they covered before specialization. In short, in the specialization step we have replaced one more general rule with two specialized rules. The first set of specialized rules covers cases such as the ***strong*** performers, thereby isolating the good performers under one rule set; the second keeps the other set of rules intact as they were before. This second set can be examined later, if found interesting.

Two points should be observed. First note that this operation of specialization on the ***strong*** performers leads to no improvement in performance at this point because the same consequent is used in the rule. Nonetheless, the rule has been honed to more accurately invoke the cases it should, and other cases have been segregated for later analysis. Second note that the **Test** for new common features is also conducted on the ***poor*** performers. The procedure is the same as for the ***strong*** ones, so it will not be repeated here. However, control is passed to the **Explain** block whether or not new common features have been found for ***poor*** performers, as indicated in figure 4.10, and as will be described momentarily.

Specialization: Store and Memory

In figure 4.10 the steps **Store** and **Memory** return the new rules to the Strategy Selector Rule Base in the KBSOS. Notice that the order the rules are stored in that rule base does not matter.

Specialization: Explain

Figure 4.10 indicates that the **Explain** module is entered under two different scenarios. The first is if no new common feature has been found for the ***strong*** performers. As explained above, the specialization module is exited for this scenario. The other situation in which the **Explain** module is called is for the case of ***poor*** performers. As indicated in figure 4.10, control passes to the **Explain** block whether or not a new common feature was found in the **Test** block for the ***poor*** performers. If a common feature was found, then the new feature (i.e., concept) is added as an antecedent to the appropriate rules, as in the case of the ***strong*** performers. If no common feature is found, no action is taken in this block. In either case, control passes to the **Repair** block because these are ***poor*** performers, and a new search strategy should be considered.

Specialization: Repair

In this block, a new search strategy is recommended. The Learner then carries out an experiment with the changed rule(s) (i.e., with a new search strategy for a consequent). If results indicate no better performance, then the specialization module is exited. If better performance is suggested, then control passes to the **Assign** block as before; the rule has been repaired and modified. We call this Rule Modification.

The decision of which search strategy to suggest as an alternative for a ***poorly*** performing case is a topic for future research. We are presently looking at the problem as a mapping of search strategy in classification space. That is, we believe a fruitful way to address the problem is to map the effectiveness of all possible search strategies as a function of the output parameters set by the KBSOS Classifier (figure 4.7). In this way one would be able to say that, for example, search strategies 3 and 7 are "adjacent" in classification space to the current search strategy, and therefore, that either of these two strategies might be attempted, given the failure of the present strategy. This mapping, which we call a strategy mapping, would reside physically in the Learner Data Base, as indicated in figure 4.4, and could itself be a candidate for further learning.

With specialization and rule modification completed in figure 4.10, direct activity now returns to the Controller of figure 4.9.

In summary of the example, we note that specialization is the vehicle by which new concepts can be added, and rule modification is the means by which new strategies can be added.

Summary

In some of our earlier research (Greenwood et al.), we suggested that machine learning could be applied usefully to the simulation optimization problem. In this chapter, we have suggested an architecture and a detailed design for how this can be done. This is the major contribution of this research.

In particular, we have defined what "learning" means in a simulation optimization setting, have shown what it is that can be learned, and have suggested that four types of learning discussed in the literature be included in a Learner. We recommend that the Learner be a knowledge-based system, and that it include a data base, a methodology base, and a rule base. We stipulate that these Learner components be directed to perform the four types of learning referred to above, namely, specialization, rule modification, parameter modification, and generalization. In addition, we have described a Controller that looks at interesting KBSOS history to specify which of the four learning types to invoke. And we have also provided detailed process diagrams to indicate how each of these learning types is carried out, when called by the Controller. Finally, we have given a detailed example illustrating the Learner architecture and flow. This example showed many implementation details as well.

Although much work remains to be done, this research initiates the discussion of learning in simulation optimization and provides a framework around which the discussion may center. In the long run, we believe that the ability to improve its performance will eventually lead to wider application of simulation optimization and thus of simulation itself.

Chapter 5: Contributions and Future Work

Contributions

Knowledge-based simulation optimization system

This dissertation further develops the idea first presented in Greenwood, Rees and Crouch (to appear) that simulation optimization can and should be subdivided into heuristic ("art") and procedural ("science") components. Each component is implemented using the most appropriate scheme: a knowledge-based system for the former, and a procedural language for the latter. The process and architecture for combining these two components into a simulation optimization system is presented. The knowledge-based system, built in VP-Expert, guides the simulation optimization process, and calls the appropriate FORTRAN programs as needed to carry out procedural aspects.

The key feature of this knowledge-based simulation optimization system (KBSOS) is that it uses a classifier and heuristics to guide the optimization. It first classifies the response surface according to several characteristics by applying procedures and heuristics to a sampling of responses from a given simulation model. Heuristics then use this characterization to select an appropriate search strategy. The search for the optimal response is carried out accordingly. As new information is obtained about the surface from runs of the simulation model, the response surface is reclassified. The search strategy remains the same or is changed according to the most recent classification, and the search continues.

The advantages of this approach to simulation optimization are twofold. First, instead of being limited to using only one search strategy (whether or not it is the most productive for a given response surface), several are offered and guidance is given for deciding between them. Secondly, the search strategy can be changed midstream if the new information obtained about the surface during the search indicates that a different strategy would be more productive.

Learner

By their very nature, heuristics cry out for refinement and improvement; this is what learning is about. The knowledge-based simulation optimization system discussed above is a perfect candidate for machine learning. This dissertation presents four types of learning that are appropriate to the heuristics used for classification and search-strategy selection. They are: specialization of rule antecedents, modification of rule consequents, parameter modification, and generalization of rule antecedents. The process and architecture for performing these types of learning are presented.

Future Work

There are many aspects of using a knowledge-based system for simulation optimization with learning that need further study and development. Below are some of our plans in this regard.

Near-term plans

The first work to be done is to generate response surfaces implicitly with a simulation model rather than utilizing an analytical expression to which error is added. Initially, this should be a simple simulation model, perhaps again with two inputs and one output. The contribution would be in developing the interface between the simulation model and the knowledge base and in "proving" that the method works on a real simulation model.

The next area of study is an extension of the first, using a more complicated simulation model with more interesting and "real world" features. This could include, for example, discrete-valued inputs (the current KBSOS assumes continuous-valued inputs), as well as a greater number of inputs (perhaps three or four instead of two). In order to process discrete-valued inputs, the shotgun and synthesis sections of the classifier -- both heuristics and procedures -- would have to be revamped. Moreover, not all search strategies can accommodate discrete inputs; strategy-selection rules will have to be modified, and perhaps additional strategies added.

With a greater number of inputs, an additional issue of import is how to allocate shotgun runs so as to avoid an exponential increase in the required number of simulation runs. Also, the synthesis portion of the classifier will have to be automated in some way. Specifically, it is desirable that neural networks be generated that have the number of inputs and outputs appropriate to the simulation models whose responses they are emulating.

A third area of work is the development of data structures to store all the information generated during simulation optimizations. This step is critical -- the Learner must have access to all history (i.e., data from each simulation optimization that has been performed by the KBSOS) in order to function.

The final near-term step is to build the complete Learner (write rules and code) and test it on the KBSOS resulting from the first two studies above. Perhaps the two simulation models above can be used to generate several sets of simulation-optimization data (by varying parameters, for example) which can then be used to test and demonstrate the Learner.

Long-term plans

There are a number of ways to develop the KBSOS with Learning further, and also ways to use it to study questions of interest. The following are a few of these ideas.

- The KBSOS can be used to study specific types of simulation models. One problem of interest is that of singularities that can occur in the output of queueing models, and how to seek optimal solutions in these cases.

- Rather than prescribing which one search strategy should be pursued at any particular time, another approach to guiding simulation optimization is to indicate which search strategies *not* to pursue. With this approach, the KBSOS would select from those strategies not disallowed, using some yet-to-be-determined criteria, such as minimum projected time to completion. Alternatively, the user could make the choice based on personal preference, familiarity, etc.

- Rule modification in the Learner involves finding more productive search strategies for specific types of response surfaces. The question then is which strategies to test when seeking a better one. One way of course is just to try all the available ones. The plan here is to develop a more reasoned manner for choosing a strategy to test -- what might be called a "nearest neighbor" approach. All the strategies available to the KBSOS -- including those which its rules do not yet recommend -- would be ranked according to productivity and applicability for each characteristic by which surfaces are classified. Then when a new strategy needs to be suggested, the characteristics for the given surface would be mapped to the rankings, and the strategy with the highest ranking overall would be tested.

Bibliography

Bengu, G. and J. Haddock (1986) "A generative simulation-optimization system," *Computers and Industrial Engineering*, 10(4), 301-313.

Berger, J. (1989) "ROENTGEN: A case-based approach to radiation therapy planning," *Proceedings of a workshop on case-based reasoning*, San Mateo, CA: Morgan Kaufmann, 218-223.

Biles, W. E. (1975) "A response surface method for experimental optimization of multi-response processes," *Industrial and Engineering Chemistry: Process Design and Development*, 14(2), 152-158.

Biles, W. E. and J. J. Swain (1977) *Proceedings: Winter Simulation Conference*, Gaithersburg, MD, 135-142.

Biles, W. E. and J. J. Swain (1979) "Mathematical programming and the optimization of computer simulations," *Mathematical Programming Study*, 11, 189-207.

Box, G. E. P. and N. R. Draper (1987) *Empirical Model-Building and Response Surfaces*, New York: John Wiley and Sons.

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), 1-45.

Brightman, H. J. (1978) "Optimizing through experimentation: applying response surface methodology," *Decision Sciences*, 9(3), 481-495.

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, ed., Palo Alto, CA: Tioga Publishing Comptany.

Daughety, A. F. and M. A. Turnquist (1978) *Proceedings: Winter Simulation Conference*, 183-193.

Daughety, A. F. and M. A. Turnquist (1981) "Budget constrained optimization of simulation models via estimation of their response surfaces," *Operations Research*, 29(May/June), 485-500.

Forsyth, R. and R. Rada, (1986) *Machine learning: applications in expert systems and information retrieval*, New York: Halsted.

Gale, W. A. and D. Pregibon (1985) "Artificial intelligence research in statistics," *The AI Magazine*, (Winter), 72-75.

Greenwood, A. G., L. P. Rees, and I. W. M. Crouch, "Separating the art and science of simulation optimization: a knowledge-based architecture providing for machine learning," IIE Transactions, to appear.

Hahn, G. J. (1985) "More intelligent statistical software and statistical expert systems: future directions," *American Statistician*, 39(1)(February), 1-8.

Hammond, K. (1989) *Case-based planning: viewing planning as a memory task*, New York: Academic.

Hammond, K. (1986) "Case-based planning: an integrated theory of planning, learning, and memory," *Ph.D. diss.*, Yale University.

Hammond, K. (1984) "Indexing and Causality: the organization of plans and strategies in memory," *Technical report 351*, Dept. of Computer Science, Yale University.

Harmon, P. and D. King (1985) *Expert Systems: Artificial Intelligence in Business*, New York: John Wiley and Sons.

Khuri, A. I. and J. A. Cornell (1987) *Response Surfaces*, New York: Marcel-Dekker.

Klimasauskas, C., J. Guiver, and G. Pelton (1989) *NeuralWorks Professional II and NeuralWorks Explorer*, Pittsburg, PA: NeuralWare, Inc.

Kolodner, J. (1988) "Retrieving events from a case memory: a parallel inplementation," *Proceedings of the DARPA workshop on case-based reasoning*, San Mateo, CA: Morgan Kaufmann, 233-249.

Koton, P. (1989) "SMARTPLAN: A case-based resource allocation and scheduling system," *Proceedings of a workshop on case-based reasoning*, San Mateo, CA: Morgan Kaufmann, 285-289.

Mead, R. and D. J. Pike (1975) "A review of response surface methodology from a biometric viewpoint," *Biometrics*, 31(December), 803-851.

Meketon, M. S. (1987) "Optimization in simulation: A survey of recent results," *Proceedings: Winter Simulation Conference*, 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, ed., Los Altos, CA: Morgan Kaufman Publishers, Inc.

Minsky, M. (1985) *The Society of Mind.* Cambridge, MA: MIT Press.

Montgomery, D. C. and V. M. Bettencourt Jr. (1977) "Multiple response surface methods in computer simulation," *Simulation*, 29(4), 113-121.

Montgomery, D. C. and D. M. Evans Jr. (1975) "Second-order response surface designs in computer simulation," *Simulation*, 26(6), 169-178.

Myers, R. H. (1971) *Response Surface Methodology*, Boston: Allyn and Bacon.

Myers, R. H. and A. I Khuri (1979) "A new procedure for steepest ascent," *Communications in Statistics, Part A -- Theory and Methods*, A8(14), 1359-1376.

Myers, R. H., A. I. Khuri and W. H. Carter (1989) "Response surface methodology: 1966 - 1988," *Technometrics*, 31(2), 137-157.

Nachtsheim, C. J. (1987) "Tools for computer-aided design of experiments," *Journal of Quality Technology*, 19(3)(July), 132-160.

Navinchandra, D. (1988) "Case-based reasoning in CYCLOPS, a design problem solver," *Proceedings of the DARPA workshop on case-based reasoning*, San Mateo, CA: Morgan Kaufmann, 260-270.

O'Keefe, R. (1986) "Simulation and expert systems -- a taxonomy and some examples," *Simulation*, 46(1)(January), 10-16.

Pao, Yoh-Han (1989) *Adaptive Pattern Recognition and Neural Networks* Reading, MA: Addison-Wesley Publishing Co., Inc.

Pegden, C. D. and M. P. Gately (1980) "A decision-optimization module for SLAM," *Simulation*, 34(1), 18-25.

Reddy, Y. V. R., M. S. Fox and N. Husain (1986) "The knowledge-based simulation system," *IEEE Software*, 3(2)(March), 26-37.

Rees, L. P., E. R. Clayton and B. W. Taylor, III (1985) "Solving multiple response simulation models using modified response surface methodology within a lexicographic goal programming framework," *IIE Transactions*, 17(1)(March), 47-57.

Remus, W. E. and J. E. Kottemann (1986) "Toward intelligent decision support systems: an artificially intelligent statistician," *MIS Quarterly*, 10(4), 403-418.

Rolston, D. W. (1988) *Principles of Artificial Intelligence and Expert System Development*, New York: McGraw-Hill Book Company.

Safizadeh, M. H. (1990) "Optimization in simulation: Current issues and the future outlook," *Naval Research Logistics*, 37, 807-825.

Shannon, R. E., R. Mayer and H. H. Adelsberger (1985) "Expert systems and simulation," *Simulation*, 44(6), 275-284.

Simon, H. A. (1983) "Why should machines learn?" *Machine Learning: An Artificial Intelligence Approach*, Michalski, Carbonell and Mitchell, ed., Palo Alto, CA: Tioga Publishing Comptany.

Slade, S. (1991) "Case-based reasoning: a research paradigm," *AI Magazine*, 12(1): 42-55.

Smith, D. E. (1973a) "An empirical investigation of optimum-seeking in the computer simulation situation," *Operation Research*, 21(2): 475-497.

Smith, D. E. (1973b) "Requirements of an optimizer for computer simulations," *Naval Research Logistics Quarterly*, 20(1), 161-179.

Smith, D. E. (1974) "A general-purpose computer program for obtaining improved simulation solutions," *Computers and Operations Research*, 1(3,4), 467-478.

Smith, D. E. (1975) *Automated RSM in digital computer simulation Volume 1 -- Program description and user's guide*, AO-A016 286, Office of Naval Research, (September).

Smith, D. E. (1976) "Automated optimum-seeking program for digital simulation," *Simulation*, 27(1), 27-31.

Velleman, P. F. and A. Y. Velleman (1988) *Data Desk Professional* 2.0, Northbrook, IL: Odesta Corporation.

VP-Expert: Rule-based expert systems development tool (1989) Orinda, CA: WorkTech Systems, Inc.

Vita

Ingrid Wessberg Moksvold Crouch was born on 31 March 1964 in Poughkeepsie, New York. She graduated from Roy C. Ketcham High School in 1981. In 1985 she graduated with a Bachelor of Science in Electrical Engineering from Virginia Polytechnic Institute and State University and was certified as an Engineer-in-Training by the Commonwealth of Virginia. She completed a Master of Science in Management Science at the same university in 1988.

Mrs. Crouch was employed by the department of Management Science as an Instructor of Quantitative Methods for one year and as a Graduate Assistant for three years. She also assisted Dr. Loren P. Rees and Dr. Terry R. Rakes with a consulting project on the effect of road traffic flows on the port of Hampton Roads, VA for the Virginia Center for World Trade.

Mrs. Crouch is a member of the Decision Sciences Institute and The Institute of Management Sciences.

Ingrid W. M. Crouch