A Methodology For Adaptive Performance Improvement Of Operating Systems

by

David Reiner and Tad Pinkerton

Computer Sciences Technical Report #377

January 1980

A Methodology For Adaptive Performance Improvement Of Operating Systems

David Reiner and Tad Pinkerton

Computer Sciences Department
University of Wisconsin - Madison

January, 1980

Abstract

This report focuses on the use of dynamic modification of operating system control parameters to improve system performance. The need for a methodology to approach this type of adaptive control is discussed, and the methodology is presented informally to acquaint the reader with the idea of real-time experimentation based on system position within a load-performance space. The methodology is then presented more formally, with emphasis on concomitant statistical techniques and possible iterative pathways within the methodology. The first two phases of the methodology serve to establish the environment for experimentation, and are followed by an experimental phase where alternative parameter settings are compared by experimenting on the live system. A productive phase then uses the results of experimentation to modify key control parameters on a periodic basis, in order to respond to fluctuations in system load and performance.

Table Of Contents

List Of Figures And Tables

I. Introduction and Problem Definition

The Domain Of A Large-Scale Operating System

Large-scale computer operating systems are generally written for an entire family of related machines, such as the IBM 360 series or the UNIVAC 1100 series. For a given machine within such a family, a variety of peripheral and processor configurations are possible. A given hardware configuration may be used in timesharing and/or batch environments, at installations whose applications range from business data processing to scientific, research-oriented computing.

At a particular site, long-term trends and changes in system load can be expected, corresponding to changes in applications, user population, user requirements, and available software. The workload also fluctuates in a periodic fashion during the week and throughout each day, as the batch load on the system varies in degree and characteristics, and as timesharing users log on and off. Even on a very short-term basis, there are significant variations in device queue lengths, memory reference patterns, and the storage allocation map (see Figure 1.1).

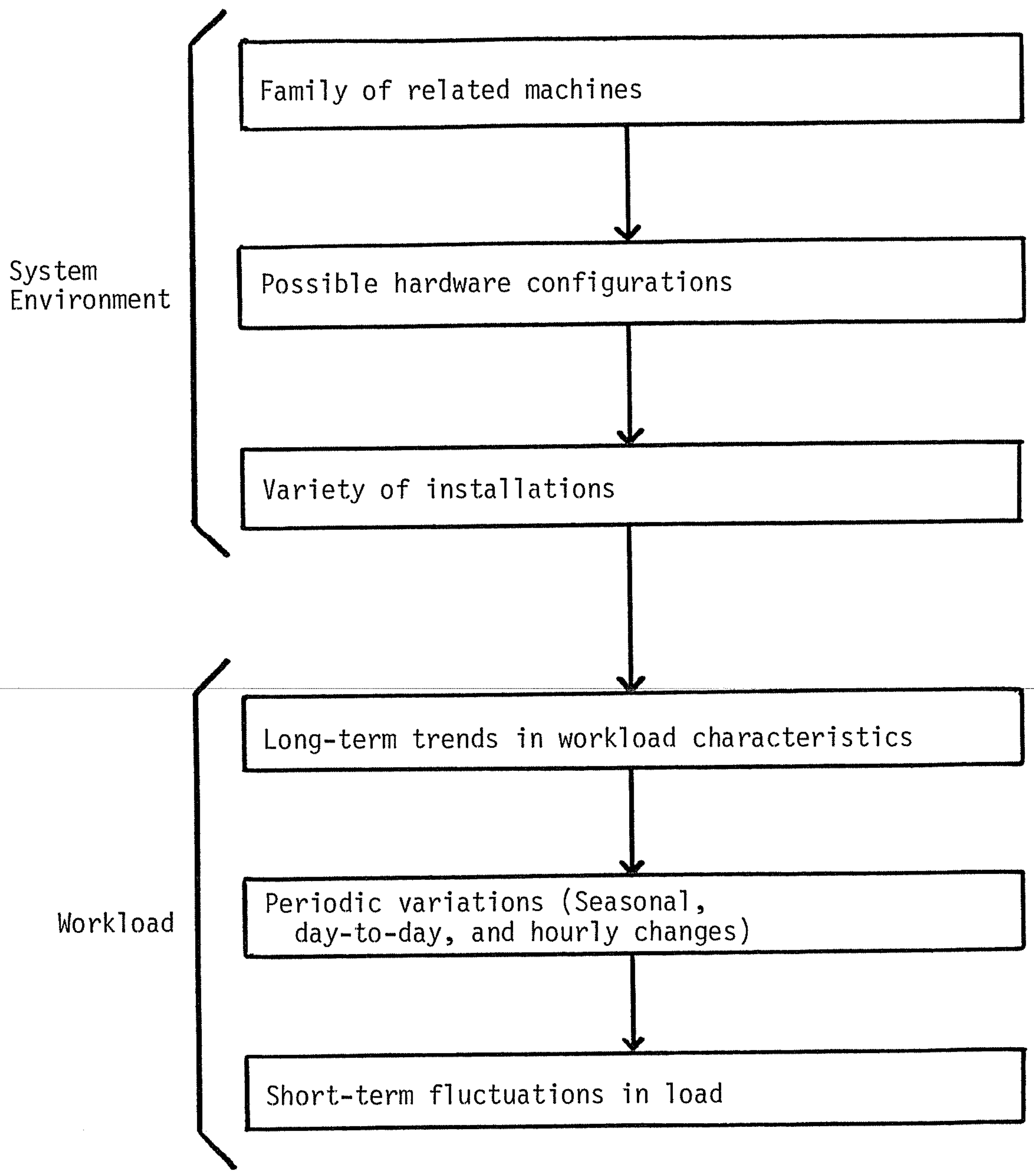
Economic considerations provide the vendor with a strong incentive for the standardization of an operating system, despite the large number of possible environments in which it must function. This is a complex and difficult task for the system designer. The flexible, generalized, and universal systems which ensue require some modification before they will yield optimal performance for a specific hardware configuration, installation, and workload. Even then, they are usually less reliable and consume more resources than a system which is custom-built for its environment and workload.

Operating System Control Parameters

There are basically three levels of operating system control parameters provided by the manufacturer which can be modified to adjust or "tune" the system to its particular environment and workload.

a) System generation parameters have their values fixed when the operating system is generated. These are not modifiable either because they have guided the generation process or because the resulting system has been shaped by their values in some permanent manner. These parameters include, for example, buffer lengths, table sizes, and the selection of permanently resident modules of the operating system.

The Domain Of A Large-Scale Operating System


System Environment
Family of related machines
Possible hardware configurations
Variety of installations
Workload
Long-term trends in workload characteristics
Periodic variations (Seasonal, day-to-day, and hourly changes)
Short-term fluctuations in load

(Figure 1.1)

b) Installation parameters are generally set into specific memory locations each time the operating system is assembled (or compiled) and linked. Examples of these parameters are the amount of buffer space available to the operating system, the amount of swapping space allocated for user programs, and the CPU quanta given to executing processes by a table-driven, multilevel feedback algorithm. Such parameters are rarely modified once they have been set. Like system generation parameters, they have been provided by the vendor as a static system adaptation to the environment.

c) Algorithmic parameters are varied by algorithms within the operating system itself (or sometimes from the console), in an effort to respond to variations in workload. Job scheduling, resource allocation, and memory management algorithms, for example, have such parameters.

It has not been customary to view operating system control parameters in terms of the above three binding times. Indeed, the distinctions between these levels of parameters are not clear-cut, and the choice of level for a particular parameter is not always made either explicitly or correctly by the operating system designers. Parameters which are static on one system may well have dynamic analogues on another, and may not exist at all on a third system.

The Potential Of Installation Parameter Modification

For a given system, system generation parameters can be changed only by building a new version of the system. Algorithmic parameters are already being altered when deemed necessary. However, there are a large number of parameters in the middle group which the operating system does not change but which could be modified. Many of these installation parameters are important in resource management and seem likely to have significant impact upon system operation, but the possibility of changing them is ignored by the operating system once they are set. Actually, any parameters which the vendor might not have considered modifiable, but which are relatively easy to access and change, can be considered to be part of this group.

While the vendor may have intended installation parameters to be static, workload variations are so great that their optimal values may change, perhaps drastically, along with workload fluctuations. It seems intuitively reasonable that dynamic modification of installation parameters has the potential to improve system performance, through improved response to changes in workload. By extending the operating system's capabilities of self-adaptation, such modification in effect enables the system to specialize itself more fully for its particular environment.

The Need For A Methodology

Unfortunately, there are a number of serious problems involved in this type of adaptive control. While a parameter may be easy enough to modify, the effects of a change may be unclear or difficult to observe. This is particularly true for large systems, where workload and resulting performance variations are sufficiently large to obscure changes in performance caused by resetting control parameters.

The wrong choice of corrections may cause the system to become unstable, oscillate, or even crash. Furthermore, frequent changes to control parameters may induce transient effects which degrade overall system performance.

Clearly, guesswork or unstructured attempts to introduce adaptive control have little chance of success with complex systems. What is needed is a methodology for approaching the problem of improving performance through dynamic modifications to installation control parameters.

Analytic And Simulation Models

Analytic and simulation models are frequently used in the analysis of operating systems, mainly in the areas of system design, system configuration or reconfiguration, and the discovery of bottlenecks and areas of poor performance. The response to the results of such analyses is typically to restructure a contemplated system, buy more equipment, modify the system workload, or change the system's generation parameters or algorithmic parameters. Altering installation parameters is not a common response.

The methodology we seek should identify and respond to those variations in system workload not generally perceived or identifiable by simulation and analytic techniques. Certainly there is no analytic function of system performance measurements on a complex system which indicates which control parameters should be modified, when and how often such changes should take place, or what the best settings might be. Furthermore, the level of improvement obtained by our methodology may be only a few percent or less, which is quite significant on a large system, but exceeds the resolution of simulation and analytic techniques.

Desirable Properties Of The Methodology

Our methodology must guide the development of a practical link between the values of system load (and performance) measurements, and the corresponding control parameter settings which optimize (or at least improve) system performance, as measured by some readily calculated (computed) evaluation function. It is desirable that the methodology be:

a) Experimental -- based on the actual system instead of a model;

b) Iterative -- to reflect growing understanding of system dynamics on the part of the experimenter;

c) Non-disruptive -- since experiments are done during production to measure and observe the real system;

d) Inexpensive -- since the resulting improvement may be small;

e) Easy-to-use -- for practical reasons;

f) System-independent -- for wide applicability and generality;

g) Robust -- for greater probability of success; and

h) Statistically sound -- so that the significance of the results can be demonstrated and verified.

Related Research

In recent years, interest in computer system performance improvement has been increasing steadily. A significant part of current research involves adaptive, learning, self-regulating, or self-tuning systems. While it is beyond the scope of this technical report to survey the current state of the art in these systems, we have listed below a number of publications which are of special interest.

Experimental approach to system performance: [BARD, 1973], [BARD, 1975], [BOX & DRAPER, 1969], [FERRARI, 1977], [FRIEDMAN & WALDBAUM, 1975], [KELLY, 1977], [KOLENCE, 1973], [RODRIGUEZ-ROSEL, 1971].

Learning and developing strategies: [FU, 1973], [KIMBLETON, 1975], [SAMUEL, 1967], [SLAGLE, 1971], [SLAGLE & BURSKY, 1971], [SMITH, 1973], [SMITH, 1977].

Adaptive or policy-driven resource allocation and scheduling: [BADEL, 1974], [BERNSTEIN & SHARP, 1971], [BLEVINS & RAMAMOORTHY, 1971], [BUNT, 1975 (1)], [BUNT, 1975 (2)], [BUNT, 1976], [BUNT & HUME, 1972], [CHANSON & BISHOP, 1977], [DOHERTY, 1971], [GECK, 1979], [KRITZINGER, KRZESINSKI, & TEUNISSEN, 1978], [MITRANI & HINE, 1977], [NORTHOUSE & FU, 1973], [POTIER *et al*, 1974], [SHARP, 1973], [SHARP & ROBERTS, 1974], [TEOREY & PINKERTON, 1972], [WILKES, 1971].

II. Development Of A Methodology For Performance Improvement

Goal Of The Methodology

As observed in Section I, optimal settings of operating system control parameters (installation parameters) are likely to vary with changes in system workload. The methodology should thus anticipate and focus on the eventual implementation of a dynamic parameter-modification policy or algorithm.

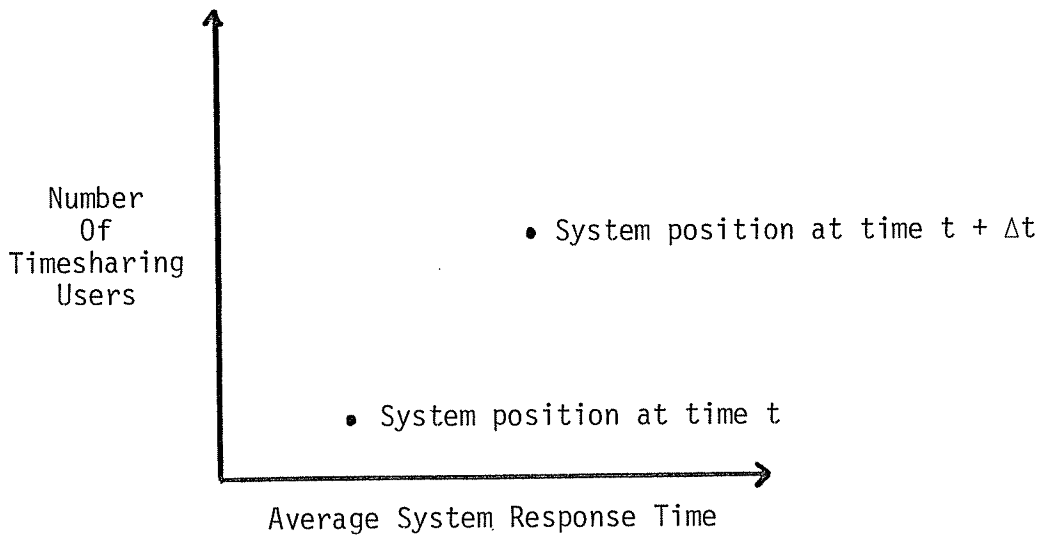
The Load-Performance Space

The use of adaptive corrections based on system workload is referred to as _feedforward_ control; when corrections are based on the system's performance under a workload, this is called _feedback_ control. _Hybrid_ control policies allow both feedforward and feedback control, and are more flexible than either policy alone. The term _load-performance space_ (_L-P space_) will be used to refer to the Cartesian product of the sets of possible values for system measurements along each of various axes (or dimensions) where system load and its performance under that load can be determined or measured. Basically, it is changes in the system's position in some load-performance space which should trigger modifications to parameter settings by hybrid control mechanisms.

As an example, consider a two-dimensional L-P space associated with an arbitrary interactive computer system S. Let one dimension be the number of timesharing users currently logged onto terminals (a load measurement); let the other dimension be the average system response time to terminal users' commands over the previous minute (a performance measurement). One might well expect that a change in system position in this space could necessitate a change to an installation parameter such as the basic CPU timeslice allocated to interactive users, in order to achieve better system performance (see Figure 2.1).

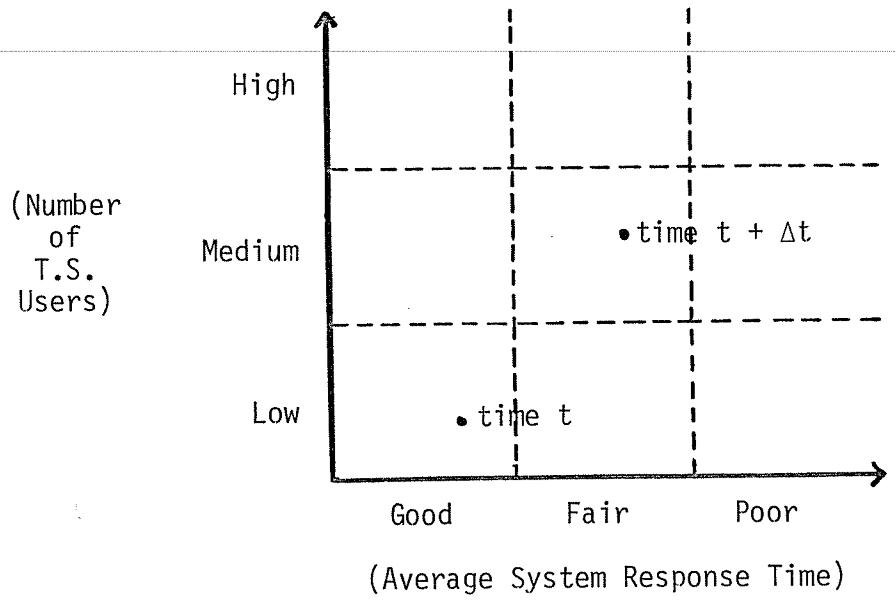
Given a system and an associated L-P space, it is neither desirable nor possible to react to every minute displacement of system position in the space. It is reasonable to partition the L-P space into a number of _load-performance regions_ (or _cases_), under the assumption that system movement from one region to another is a necessary and sufficient change to require modification of control parameters. Ideally, regional contours should follow roughly the lines of demarcation between different optimal policies or parameter settings. The problem, of course, is that such regional contours, or divisions between cases, are unknown and must be determined through experimentation, along with corresponding parameter modifications.

Two-Dimensional Load-Performance Space Associated With A System S


Number Of Timesharing Users
• System position at time t + Δt
• System position at time t
Average System Response Time

(Figure 2.1)

One Possible Case Space Based On The Load-Performance Space Associated With S



The grid defines nine regions, or cases.

(Figure 2.2)

A convenient method for dividing an L-P space into cases, attractive because of its simplicity, is to partition each axis of the L-P space into a small number of intervals. For example, the number of timesharing users on a system may be characterized as "high", "medium", or "low", and this will be the only differentiation of system position changes in L-P space based on this axis. The regions of the divided L-P space (or *case space*) are defined by the grid resulting from the partitions of the axes (see Figure 2.2).

One must strike a balance between too many regions and too few. Having too great a number of regions complicates experimentation, and casts doubt on the stability of comtemplated corrections, since the system will not stay in any one region for very long. Having too few regions may obscure system movements in L-P space which presage and ought to invoke important corrections to control parameters. The evident conclusion is that partitioning the L-P space to form the case space should be an iterative process in our methodology, reflecting the results of continued experimentation.

The Evaluation Function

In order to measure performance changes, a performance metric is required. Such a metric, or *evaluation function*, is not *a priori* well-determined, and must be to a large degree a policy function. The reason for this is that some of the performance measures available, such as system throughput and individual response time, cannot be simultaneously optimized. While the choice of evaluation criteria may be relatively easy, their relative weightings as components of the evaluation function will require careful consideration of both policy and technical aspects.

For example, a good evaluation function (sometimes called a *figure-of-merit* function) should distinguish between a poorly-balanced system and an under-utilized (lightly-loaded) system [STEVENS, 1975]. The former condition is cause for alarm, while the latter is inevitable on occasion.

Structure Of The Methodology

Many choices and decisions encountered within the performance improvement procedure must be made on insufficient information and data. These include which control parameters to modify, what dimensions to choose for the L-P space, how to divide that space into regions to form the case space, how often to modify control parameters, and so on.

sure that choices which are somewhat arbitrary make good use of the information which is available at the time, and are made in a sensible order. Furthermore, the methodology must permit, define, and encourage the use of iterative pathways along which the experimenter can go back and reconsider critical decisions and key choices in the light of further experimentation and analysis.

Since experimentation will probably show variables and measurements to be interrelated in unexpected and complex ways, an early part of the performance improvement procedure should be an attempt to select a small number of uncorrelated variables as L-P space dimensions and as evaluation function components, to make results clearer and easier to obtain.

The Duration Of Experimentation

It is difficult to discover the links between measurements and optimal control parameter settings when the effects of changing those parameters are obscured by workload fluctuations. The main weapon that the experimenter has against this problem is to observe the sytem over a _long period of time_ to accumulate a large number of observations. This provides a powerful statistical tool for data analysis, and makes it possible to distinguish performance changes induced by control parameter modifications from those due primarily to workload fluctuations.

Even when a table of control parameter modifications (the _M-table_) corresponding to regions of the L-P space has been experimentally determined, and the system is no longer being actively investigated to discover further adaptations, a background program may still experiment periodically (or on a time-sliced basis) to discover and compensate for long-term trends.

III. A Formal Description Of The Methodology

Definition Of Notation

Let S be the system or subsystem to be considered.

Let $C[S] = \{C_1, C_2, \ldots, C_q\}$ be the set of modifiable control parameters which affect the operation of S.

Let $P[S] = \{p_1, p_2, \ldots, p_r\}$ be the set of readily accessible system performance functions characterizing the status of S in terms of load and performance.

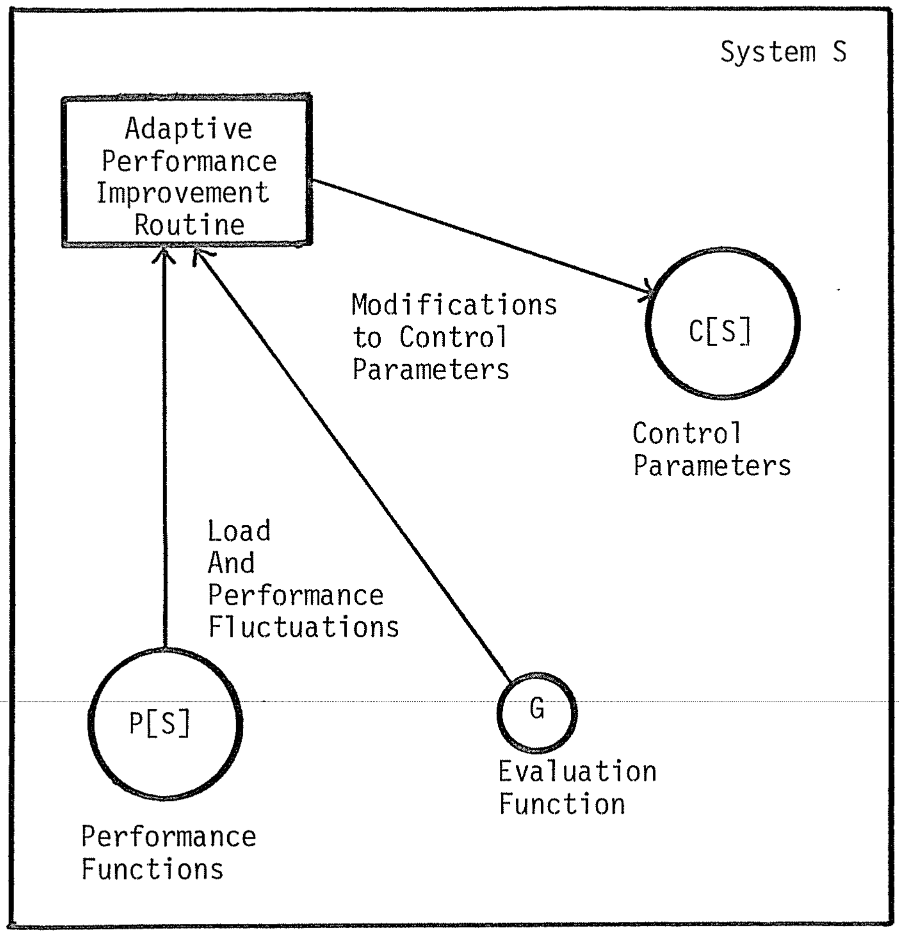
Let G (the evaluation function, or "goodness" function) be a linear combination of certain of the functions in P[S].

Statement Of Objective

By observing the behavior of S, we would like to learn how to respond to system load and performance fluctuations (characterized by the functions in a subset of P[S]), by modifying some of the control parameters in C[S], so as to improve the performance of S as measured by G (see Figure 3.1).

As shown in Figure 3.2, the methodology is divided into four phases.

Objective Of The Methodology


System S
Adaptive Performance Improvement Routine
Modifications to Control Parameters
C[S]
Control Parameters
Load And Performance Fluctuations
P[S]
Performance Functions
G
Evaluation Function

(Figure 3.1)

The Four Phases Of The Methodology

$$\begin{tikzcd}[row sep=large]
{} \arrow[d] \\
\text{\underline{Phase 0}: Establishment of System Environment} \arrow[d] \\
\text{\underline{Phase 1}: Establishment of Experimental Environment} \arrow[d] \\
\text{\underline{Phase 2}: Experimental Phase} \arrow[d] \\
\text{\underline{Phase 3}: Productive Phase}
\end{tikzcd}$$

(Figure 3.2)

Phase 0: Establishment of System Environment

0.1 Select S.

0.2 Determine C[S].

0.3 Determine P[S].

Phase 1: Establishment of Experimental Environment

1.1 Select m modifiable control parameters from C[S]. With no loss of generality, let $C = \{C_1, C_2, \ldots, C_m\}$ be this set.

1.2 Observe the system S for a period of time to collect representative values of the load-performance functions in P[S].

1.3 Determine the pairwise correlations among functions in P[S], from the data collected in Step 1.2. The results will be helpful in Steps 1.4 and 1.5. If it turns out, for example, that the correlation matrix shows p_i and p_j to be very highly correlated, then one of them can be predicted as a linear function of the other. This does not imply that p_i is somehow "responsible" for p_j, or p_j for p_i, but only that the relation between the two can be thought of as linear. (See p. 619 of HAYS, 1973, for discussion of this point.) When this is the case, then it is unnecessary to use both p_i and p_j to characterize load or performance, or as components of an evaluation function.

Another approach to determining dependencies among "independent" variables is principal-component analysis, described in BARD & SURYAMARAYANA, 1972, which uses eigenvectors of the correlation matrix to determine which variables can be solved for in terms of the others. However, the referenced analysis was used only as a general guide for detecting dependencies, and the authors relied heavily on knowledge of the system and direct study of the correlation matrix.

In principal-component analysis, no particular assumption about the underlying structure of the variables is required. Another approach, classical-factor analysis, is based on the assumption that some underlying regularity of the data is responsible for the observed correlations. As described in NIE _et al_, 1970, this technique relies on finding a small number of common determinants which will account for all or most of the observed relations in the data.

It is up to the experimenter to choose the level of sophistication of his search for dependencies among the functions in P[S]. However, much can be learned from simple inspection of the correlation matrix, and it is decidedly advantageous to select functions which have intuitive meaning for the observer, rather than dealing with factors whose relationship to the system S is not clear.

1.4 _Construct the evaluation function G, using decision analysis techniques to select its components from P[S] and establish their weights_. G is a _policy_ function, and it is very helpful to have formal techniques which prescribe how decision makers "should think systematically about identifying and structuring objectives, about making vexing value tradeoffs, and about balancing various risks..." [KEENEY & RAIFFA, 1976].

An additive evaluation function (e.g., $G = \sum_i w_i p_i$) is aesthetically more pleasing and easier to analyze than one with interactions among the components. For G to be additive, it is necessary and sufficient to establish mutual preferential independence among its components. This is satisfied, according to Keeney and Raiffa, if our preference for the various levels of a component is independent of the levels of the other components. For example, if we always prefer short terminal response time to long terminal response time, regardless of the percentage of CPU utilization, and always prefer a high to a low percentage of CPU utilization, regardless of the terminal response time, then response time and CPU utilization are mutually preferentially independent, and our preference structure can be expressed by an additive value function whose components are response time and CPU utilization.

Keeney and Raiffa describe techniques for arriving at component weights once mutual preferential independence has been determined to hold. These may be applied for one decision-maker or several. Delphi techniques [DALKEY, 1972] can also be used to elicit the components and weights of an evaluation function from a group of experts.

1.5 Select n orthogonal (as much as possible) functions from P[S]. Let P_i represent the range of p_i. With no loss of generality, let $P = (P_1, P_2, \ldots, P_n)$. "Orthogonal" is used in the sense of "uncorrelated," as discussed in Step 1.3.

The set of all possible values for the vector P will be called the load-performance space of S (L-P space of S), and will be written S/P. Where "X" represents the standard Cartesian product of sets, we have:

$$S/P = P_1 \text{ X } P_2 \text{ X } \ldots \text{ X } P_n$$

We expect k of the p_i to be load functions, and n - k to be performance functions, for some k between 0 and n. Note that if we base adaptive control mechanisms on system position in S/P, then:

$k = 0$ ==> feedback system;

$0 < k < n$ ==> hybrid control system;

$k = n$ ==> feedforward system.

S/P is n-dimensional, and is the union of the k-dimensional load-subspace of S and the (n-k) - dimensional performance subspace of S.

1.6 Using the data from Step 1.2 (and making additional observations of S if necessary), partition each P_i, $i=1,2,\ldots,n$ into a small number L_i of intervals. With luck, the data will be multi-modal, and a natural partition will suggest itself for each dimension. If not, one can make the intervals of equal size as a first approximation. (See page 32 for comments on later reconsideration of these partitions.)

For each P_i, number the intervals 1, 2, ..., L_i, and define $\bar{P}_i$ as the interval number corresponding to the value of p_i. (Recall that P_i is the range of p_i.) The case space of S is defined as:

$$S/\bar{P} = \bar{P}_1 \text{ X } \bar{P}_2 \text{ X } \ldots \text{ X } \bar{P}_n \quad .$$

The elements of $S/\bar{P}$ are called <u>cases</u>. The number of cases is $L_1 L_2 \ldots L_n$. When the system is observed to be at a certain location in S/P at a given moment in time, S is said to be <u>in</u> the case which includes that location. S is in one and only one case at a given time, since $S/\bar{P}$ is a partition of S/P.

1.7 [This step is omitted the first time through Phase 1, and will usually be entered from Step 2.5. It is discussed at the end of this chapter on page 32.]

Restatement of Objective

For each case in $S/\bar{P}$, we wish to discover the modifications to C which maximize G (or at least improve it significantly). The table of such modifications, arranged by case, will be referred to as the <u>modification table</u>, or <u>M-table</u>.

Phases 0, 1, and 2 of the methodology are concerned with the development of the M-table; Phase 3, with its productive use.

Phase 2: Experimental Phase

2.1 <u>For each case in $S/\bar{P}$, determine a number of possible test values for C (or modifications to C)</u>. These may or may not be case-dependent, but should include the current parameter setting (C_0) as a control. The test values represent possible entries for the M-table, and should be carefully chosen. For example, if C_i may range between A_i and B_i, then choose a number of test values spaced equally along the interval $[A_i, B_i]$. The experimenter may wish to be more cautious at first, and pick test values in a narrower range than the entire interval $[A_i, B_i]$, perhaps deviating only slightly from the current setting(s). For some control parameters, it may be true that no effects on system performance can be observed except in a very small part of the parameter's range, often at or near one of the extreme settings. If C is multidimensional, then factorial design techniques [KEPPEL, 1973] are appropriate to test interactions among its components.

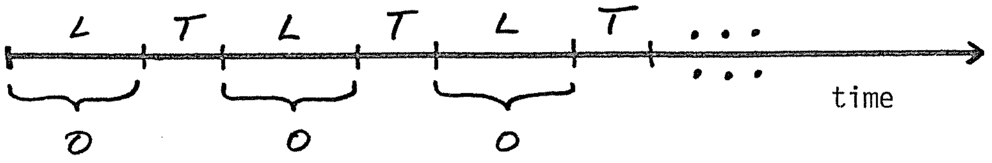
Some tradeoffs are involved in the choice of test values. Failure to effectively bracket the range of a particular C_i may result in missing an optimal setting. Yet exhaustive testing is very time-consuming, especially using a factorial design on several factors (components of C). It is possible to compromise by starting with relatively few test values, then refining or augmenting the selection of test values based on the trends observed during testing.

This ties in with the need for caution -- it may not be desirable to test the full range of possible control parameter settings from the start without having a good idea what the effects of small changes will be.

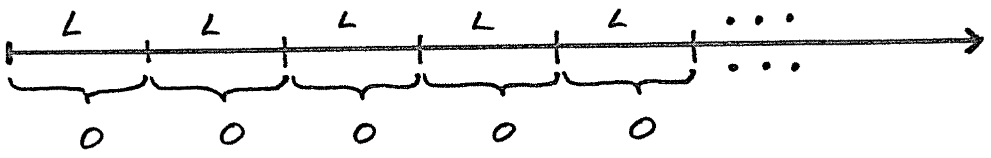
2.2 Determine the parameters which govern the investigative process. These include the length of time over which each set of test values for C will be assessed (the learning interval), and the length of time after C has been modified before a new learning interval commences (the transient interval). In Phase 3, when the emphasis shifts from investigation to production, the transient interval will not exist, and the learning interval might more properly be referred to as the "intra-observation and modification" interval (see Figure 3.3).

Learning And Transient Intervals

Phase 2: Experimental Phase



Phase 3: Productive Phase



L = learning interval
T = transient interval
O = observe system; modify control parameters at end

(Figure 3.3)

There are no absolute standards to direct the choice of interval lengths, but some tradeoffs should be considered. The learning interval must be long enough to enable stability of the observed system performance, yet short enough to catch high-frequency fluctuations. The transient interval must of course be long enough to fulfill its primary function -- to allow transients induced by control parameter modifications to die out, yet it must be short enough so that the observation process remains efficient. The nature of the control parameters being modified is the most important factor in determining the length of the transient interval. If these parameters affect the memory map, for example, it may take a minute or more for the system to stabilize under a new policy.

One more parameter which affects the process of investigation is the order in which the various test values from Step 2.1 should be tried, for any given case. Statistical considerations dictate a random method of selection among the alternatives. In particular, if test values are drawn randomly without replacement from the possible choices until all alternatives have been tried, and this method is reused repeatedly, then later comparisons of the alternatives can be based on "balanced design" techniques. In other words, the experimenter will have about the same number of observations of any given test value for each case.

2.3 Observe system behavior (as reflected in G) under periodic modification by M-table test values from Step 2.1, using the parameters discussed in Step 2.2. The long periods of time necessary for observation need not involve the experimenter directly or continuously. Much of the work can be done by low-overhead programs which periodically monitor the system (at the end of each learning interval) and try different control parameter test settings and time intervals between modifications. These results may be spooled to tape or disk files for later analysis. When experimenting over time, it is preferable to switch fairly rapidly among the possible parameter settings being studied, to further minimize the effects of load fluctuations [BARD, 1973].

2.4 For each case in $S/\bar{P}$, pick the "best" M-table entry from the results of Step 2.3. It is a useful preliminary step to examine the components of G separately to make sure that the control parameter changes have had some effect on the system, particularly as certain components of G may be inversely related, and changes to them might cancel each other out. After that, selecting the test value (or policy, as it will be referred to in the following discussion) with the highest mean observed G value is intuitively the easiest way to select the "best" M-table entry for a given case. It is appropriate to have some kind of statistical confirmation that this policy is better than the standard setting C_0 with, say, 90% or 95% probability. A number of statistical approaches can be used (many of which depend on readily-available "canned" statistical programs):

a) Compare policies using a non-parametric test such as the rank-sum test or the sign test (see pp. 280-300, DIXON & MASSEY, 1957). There are two problems here. The first is that we are ignoring the actual values of the data to concentrate on rankings (or signs of differences). This is not _wrong_, but only makes partial use of our arduously-gathered observations. The second problem is that such tests are very sensitive to the independence of the observed data. We have control over the choice of policy within a given load-performance case, but not at all over case transitions. In other words, we experiment in a given case when we are forced there (perhaps by other factors not being considered), rather than when we wish to experiment there. However, randomizing the policies can counteract this effect to a certain extent. It is unavoidable that the total number of observations will vary from case to case, since cases are not equally likely to be visited.

b) Compare policies using a t-test. This improves on a) since we are looking at actual data values of the evaluation function G. A disadvantage is that the underlying distributions which we have sampled must be assumed normal. This can be checked (with some effort) by the Chi-square test, the Kolmogorov-Smirnov test, or graphically using probability plots (see p. 116, RYAN _et al_, 1976).

c) Compare policies using the ranking and selection methodology described in MAMRAK & DERUYTER, 1977. This is an iterative methodology which selects the best policy by a two-step data-gathering procedure, where data from the first step is analyzed to determine how many additional observations are necessary to guarantee that the probability of a correct selection will be at least P^*, a value chosen by the experimenter. In addition, the experimenter must specify the magnitude of the difference that should be detected between the best and second-best means. As Mamrak and De Ruyter point out, this is a desirable feature: if policies do not differ by much, large samples of data are required to determine the best one. From the standpoint of looking for improved rather than optimal policies in a fairly short period of time, experimenters should not require too great a difference between the top two policies.

Incidentally, the authors discuss a number of techniques for transforming data which is nearly exponentially distributed into a more symmetric and more nearly normal form. They also deal briefly with the problem of reducing serial correlation of data observations.

d) If the observations are balanced (about the same number for each policy in each case), then a standard analysis of variance (ANOVA) program can be used to disprove the null hypothesis that the policy means of the observations are actually all equal. If this can be disproved, then the experimenter knows that there has been some effect on performance, and can use a multiple comparison test (such as the studentized range test [MILLER, 1976]) to get groupings and find out which policy or policies are best. ANOVA programs do exist which accept unbalanced observations, but they are tricky to use and are not recommended [BOX et al, 1978]. Francis has shown that several widely-used ANOVA programs which claim to be able to handle unbalanced data often yield misleading answers, depending on the order in which variables are considered [FRANCIS, 1973].

e) Compare policies by multiple linear (or higher order) regression of observations (evaluation function values) on the n load-performance functions $p_1, p_2, \ldots, p_n$ (the dimensions of the L-P space of S). See RYAN et al, 1976, for a description of a simple multiple regression program (in MINITAB). For each policy, this yields estimates of w_i where

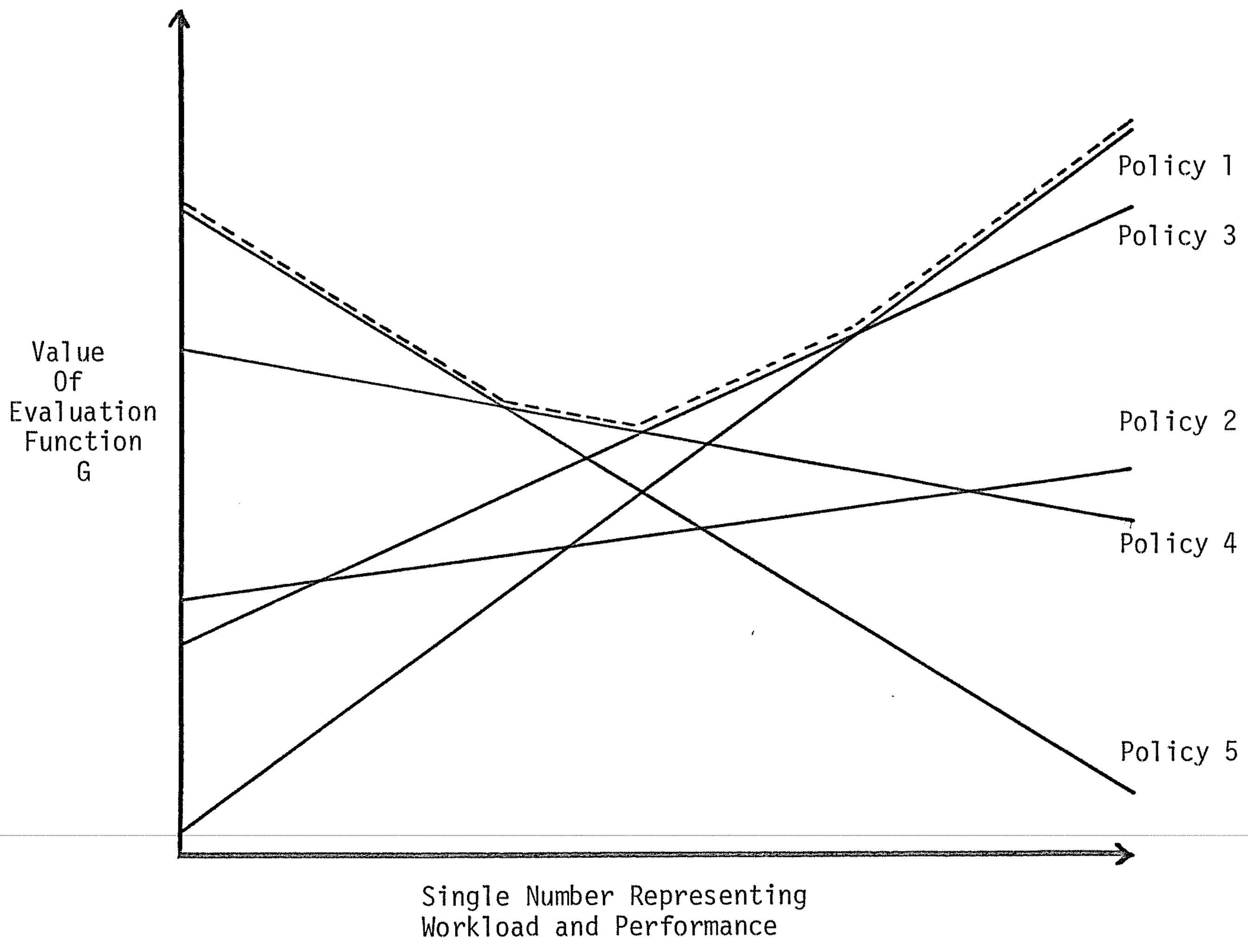
$$G = \sum_i w_i \, p_i + \varepsilon$$

Policies can then be compared for any given case using these results. If the coefficient of determination (commonly referred to as R^2) from the regression is low, however, the results are not very useful or significant. If R^2 turns out to be fairly high, then we can more or less predict performance for a given policy based on actual values of the p_i, as opposed to the other methods which used a somewhat artificial stratification (or "caseification") to characterize load and performance. In other words, we are looking at the L-P space instead of the case space.

This method, if it can be made to work, is useful as an aid to understanding how the p_i's (load-performance characterizers) affect performance (the evaluation function G) for each policy, and can also serve as a check on the stratification involved in the case space. A somewhat similar approach was used in FRIEDMAN & WALDBAUM, 1975, where the authors used regression equations relating system performance variables to variables describing the uncontrolled workload and system modifications, to determine the effect of these modifications.

f) Using a single number to represent workload and performance as defined by the p_i, compare policies by regressing this single value against the evaluation function for each policy. Graphing the regression lines for each policy on a single graph (see Figure 3.4) will make it easy to see which policy is best where. The two main problems here are: 1) it is difficult (and probably misleading) to represent all of the p_i's with one number; and 2) the confidence intervals for the regression lines may be so wide (because of noisy data) that conclusions are impossible.

Regression Lines For Each Policy



----------- = best policy contour

(Figure 3.4)

As in Step 1.3, it is up to the experimenter to select a statistical method to support his choice of the "best" policy for each case. A useful reference is GIBBONS _et al_, 1977, which gives a rigorous but readable treatment of ranking and selection procedures from a statistical point of view.

2.5 _Estimate the potential performance improvement associated with the M-table selected in Step 2.4._ At this point, it is of interest to the experimenter to know if the division into cases which defined the case space from the L-P space is reasonable. One would expect neighboring cases to have somewhat related results (policies) in the M-table. This is a good argument for designing a set of policies on which a _metric_ exists. If different policies are incommensurable, then there is no test to indicate that the original "caseification" was reasonable.

To estimate the potential performance improvement associated with the M-table, we calculate $\hat{\pi}$. $\hat{\pi}$ (S/$\bar{P}$; G; M-table) is defined as:

$$\frac{\sum_{i=1}^{j} (\bar{G}_{best}(i) - \bar{G}_{old}(i)) \cdot V_i}{\sum_{i=1}^{j} V_i} \cdot 100\%$$

where the cases in the M-table are assumed to be numbered 1, 2, ..., j, and where for i=1, 2, ..., j

a) V_i is the number of visits observed in case i;

b) $\bar{G}_{best}(i)$ is the average observed G value in case i for the policy (modifications to C) chosen as best in Step 2.4; and

c) $\bar{G}_{old}(i)$ is the average observed G value in case i for C_0.

Although it seems like the natural definition, if we use b) as stated above, there will be a statistical bias introduced into the calculation of $\hat{\pi}$. Instead of just taking the average G value for the best policy (call it C^*) in case i, we should take the average G value over the group of policies which are not statistically significantly different from C^*, although their average G values may have been slightly smaller. To give a more concrete example of this point, suppose that policies numbered 7, 8, and 9 are all better than C_0, but not significantly different from each other. We could then use the average of all of their G values, rather than the average G value for just policy 8, say, if it happens to be slightly greater than those of policies 7 and 9. In this way, we avoid overestimating $\hat{\pi}$.

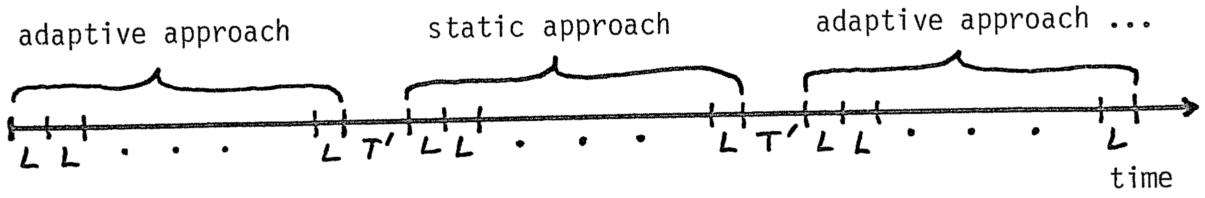
$\hat{\pi}$ is a very useful number. If the relevant raw data has been retained, two differently-structured case spaces (and their associated M-tables) may be compared in terms of their $\hat{\pi}$ values. (See Case Space Restructuring, page 32.)

2.6 Observe the system again to obtain a more accurate estimator of potential performance improvement which takes into account the effect of transients generated by frequent switches to control parameters. Up until this point, G has been measured only after intervals of time have "cushioned" the effects of any transients generated by switches in control parameter policies. In practice, there may or may not be some degradation to G during those transient intervals.

$\hat{\pi}$, defined in Step 2.5, is an estimator for π, which we define as the performance improvement which will be obtained by actually using the M-table in practice (see Phase 3 below). Obtain a more accurate estimator $\hat{\hat{\pi}}$ of π in the following manner:

Observe the system S again, alternating between setting control parameters to C_0 (the static approach) and using the M-table entries to set C (the adaptive approach). Allow each of these two approaches to run for some time (say ten times the original learning interval L), but switch between them often enough to eliminate most of the effects of workload variation. Allow time (T') for transient effects to die down between the static and adaptive approaches, of course, but measure G for the entire time interval for which each approach is in effect. In short, as shown in Figure 3.5, the transient interval T of Step 2.2 is set to zero.

Measuring Data To Calculate $\hat{\hat{\Pi}}$



(Figure 3.5)

We define $\hat{\hat{\pi}}$ (S/$\bar{P}$; G; M-table) as

$$\frac{\bar{G}_{adaptive} - \bar{G}_{static}}{\bar{G}_{static}} \cdot 100\%$$

where $\bar{G}_{adaptive}$ is the average G value observed while the adaptive approach is in effect, and $\bar{G}_{static}$ is the average G value observed while the static approach is in effect. Clearly, the experimenter would prefer $\hat{\hat{\pi}}$ to be positive and as large as possible.

If a synthetic, repeatable workload is available which is representative of overall system activity, then the adaptive and static approaches may be compared during system test time on the basis of their performance under this workload. There is no need to swap between the two approaches; since the workload is repeatable, they may be run separately for a predetermined amount of time, or until all batch jobs and timesharing requests have completed.

2.7 Attempt local optimization within each case. The M-table entry associated with each case may be good but perhaps it can still be improved on. Several techniques are available which concern themselves with gradual improvement or optimization of a control parameter (or parameters, since C may be multidimensional).

As Fu remarks [FU, 1973], there are a number of numerical methods such as stochastic approximation (see WASSAN, 1969) which depend on estimation or successive approximation of the unknown quantities of a function which represents the process under study. However, in very complicated learning situations, when the underlying mathematical model is impossible to formulate, these methods should be rejected in favor of more heuristic approaches. Kimbleton [KIMBLETON, 1975] concurs, commenting that since probabilistic approaches cannot adequately reflect system complexity, and since deterministic (mathematical programming) techniques cannot conveniently deal with random phenomena, a heuristic approach is often best. In addition, sequential search plans, where future experiments are based on past outcomes, are preferable to simultaneous search plans, where every experiment is specified before any results are known [WILDE, 1964].

Briefly, here are a few procedures which seem likely to be helpful for local optimization within cases, although most of them were designed with some type of global optimization in mind.

a) Evolutionary Operation [BOX & DRAPER, 1969] consists of trying a cycle of slight variants about the current parameter setting, for one or two key parameters. Although this technique was developed mainly for chemical works processes, it is applicable here. After a period of experimentation, the experimenter can:

1) adopt a new parameter setting;

2) explore a certain favorable direction;

3) modify the variants attempted;

4) modify the ranges through which the parameters are allowed to vary; or

5) try different control parameters.

b) The Steepest Ascent Procedure [MYERS, 1976] is a method whereby the experimenter proceeds sequentially along the path of maximum increase in response. Some possible problems are: 1) only one of a number of peaks may be approached; and 2) the procedure of steepest ascent is not invariant with respect to scaling of the variables, but depends on the choice of units (see BUEHLER et al, 1961). A good example of the use of steepest ascent is given in BOX & WILSON, 1951. Also see BUEHLER et al, 1964, for a discussion of the method of parallel tangents, and CARTON et al, 1973, for comments on gradients and other classical edge-detection techniques.

c) The Method of Ridge Analysis [DRAPER, 1963] deals with situations such as rising ridge and saddle point systems, where no optimum has yet been discovered and further analysis or experimentation seems to be called for. This method resembles the steepest ascent method, with one important difference. Steepest ascent is used before canonical analysis (which attempts to translate the axes of the original response surface so that they correspond to the principal axes of the contour system, with the introduction of new variables as necessary). Ridge analysis is designed to be used after canonical analysis as a tool to aid the interpretation of the existing response system (see p. 96, MYERS, 1976).

d) The Experimental Approach [BARD, 1975] is proposed as an alternative to stochastic approximation, which Bard feels requires too many observations, and to evolutionary operation, which he feels involves too detailed an examination of the response surface. This method assumes that system response is a function f of the modifiable system control parameters (plus an error term). Using a composite, rotatable, balanced design, a number of possible control parameter vectors are selected, and the system is run to measure response for each vector. The function f is approximated by another function F, which is continuous, approximates the observed data, and has an easy-to-evaluate, closed analytic form. After F has been constructed, a non-linear programming routine is used to find a parameter vector which maximizes F within a suitable region. With this vector as a starting point, the process is repeated.

Bard counsels small changes on a live system, and rapid switching among parameter settings to average out random variations due to workload fluctuations (see BARD, 1973).

e) _Heuristics and Learning Methods_ borrowed from the realm of Artificial Intelligence may be of some use in improving control parameter settings, although they are rarely employed in this context. Rote learning and generalization [SAMUEL, 1963] are helpful, if elementary, techniques. As described in SLAGLE & BURSKY, 1971, a learning program can supply the rest of the system with functions for estimating how the merit of an untried goal depends on the features of that goal. If untried goals are equated with possible new parameter settings, such evaluation functions could be of some worth. Several experimenters (see SMITH, 1973, and SAMUEL, 1967) have written learning programs which rely on tables of adaptively-determined weights indicating the advisability of various strategies. This could be a useful approach, since parameter settings are, in effect, strategies.

2.8 _Modify the M-table, if possible, to reflect a global strategy based on the goodness surface (average G value) above the case space of S._ While Step 2.7 was concerned with local optimization within each case, some global optimization may now be possible. It is desirable to force the system into regions (or "peaks") of high performance, ideally via a trajectory which maximizes the integral of G over time. Such a route would avoid "valleys" of low performance, and concentrate on regions where performance is generally good.

It is likely that any system movement which can be forced (or strongly encouraged) will be in the performance subspace of S, although it is not inconceivable that system position in the load subspace might be affected.

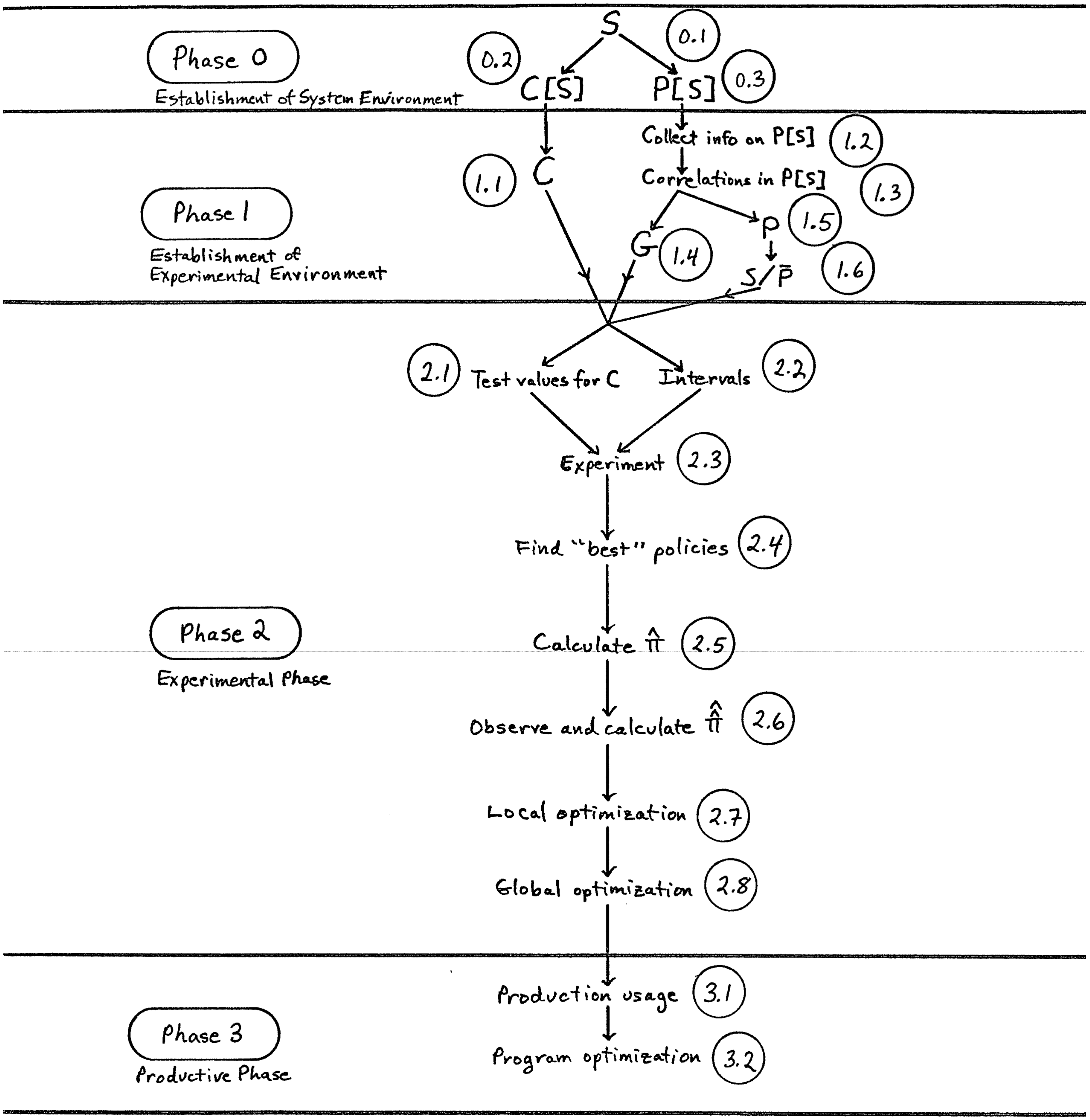
The experimenter must keep track not only of the average G value associated with a given control parameter setting in a case, but also of the vectors in L-P space representing the change in system position induced by the parameter setting, in order to accumulate information on how to urge the system in a given direction. As before, one major problem crops up -- distinguishing between random fluctuations in position and those due to parameter changes. Observation over time may yield sufficient statistical evidence to reach some tentative conclusions, but this is a thorny problem.

Phase 3: Productive Phase

3.1 Continue to modify C using the M-table at intervals of length L. During a small percentage of the time, repeat the experiments of Step 2.3, in order to detect long-range changes in the patterns of system load and performance which might require modifications to the M-table. Some of Step 2.6 may also be repeated.

3.2 Reduce the computational and space requirements of the program which modifies parameters using the M-table. During Phase 2, general code may have been necessary for flexibility of experimentation, but when results are known and an adaptive parameter-modification scheme is in full operation, specialized code and compacted tables may be substantially more efficient.

The Methodology So Far



(Figure 3.6)

Case Space Restructuring

The ideal division of the load-performance space (S/P) into cases should follow the lines of demarcation between optimal (or improved) parameter settings (or policies). Unfortunately, these policy boundaries are not known before experimentation takes place in Phase 2. Once the results of experimentation are available, it may be advisable to restructure the case space ($S/\bar{P}$) to reflect this additional knowledge about system dynamics. Such restructuring should be considered a temporary return to Phase 1, since it affects the experimental environment.

1.7) Restructure the case space $S/\bar{P}$, using the results of the system testing and experimentation performed in Step 2.5. Before restructuring can take place, it is essential to have a metric with which to compare different policies. When only one parameter is being modified, a metric is easy to define; when several parameters are altered, the situation is more complex.

If policies in adjacent cases seem to differ by a great amount, this is a danger sign. Recall that system movement from one case to another is considered a necessary and sufficient change to require control parameter modification. If the modification is large, then perhaps there should have been an intermediate case, or perhaps the regions should have been smaller in that neighborhood of the L-P space. This is especially true if the system S passes through the neighborhood frequently, for then performance will be very sensitive to any control parameter modifications associated with the cases in that neighborhood.

Conversely, if policies in adjacent cases differ by very little or not at all, this argues strongly that a change in system position from one case to the other is meaningless as far as changing parameter settings. This is especially true if the system S passes through the neighborhood infrequently, since control parameter modifications in that neighborhood are likely to have little effect on performance anyway.

An example will help to clarify this point. Suppose that we are modifying a single control parameter which can take on integral values between 1 and 80, inclusive. Let the metric for comparing two parameter settings be the absolute value of their difference. Referring back to Figure 2.2, suppose that the best parameter settings found for each case are as shown in Figure 3.7.

Intuitively, this suggests that:

a) Cases (Good,Low), (Good,Medium), and (Fair,Low) should be combined;

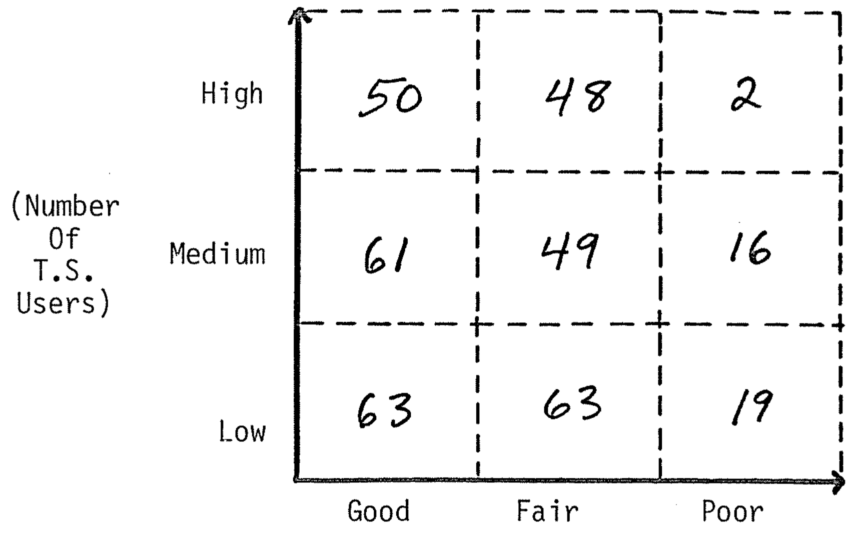
b) Cases (Good,High), (Fair,Medium), and (Fair,High) should be combined; and

c) The adjacent case pairs (Fair,High)-(Poor,High), (Fair,Medium)-(Poor,Medium), and (Fair,Low)-(Poor,Low) have such widely divergent control parameter settings that a finer regional grid in the neighborhood of Fair and Poor throughput might yield some helpful intermediate values of control parameters.

This new view (restructuring) of the case space is shown in Figure 3.8.

The experimenter's intuitive understanding of potential case space restructuring can be supplemented and clarified by the techniques of cluster analysis. These provide a formal approach to grouping together cases for which similar optimal parameter settings have been found. HARTIGAN, 1975 is a useful reference on clustering algorithms. One point Hartigan makes is that "clear-cut and compelling clusters ... require an explanation of their existence and so promote the development of theories ..." Observation of experimental results and the application of clustering techniques to the case space and its associated M-table may render a "black box" system less mysterious, at least in terms of understanding the effect on the system of the control parameters being modified.

Parameter Settings Associated With A Case Space

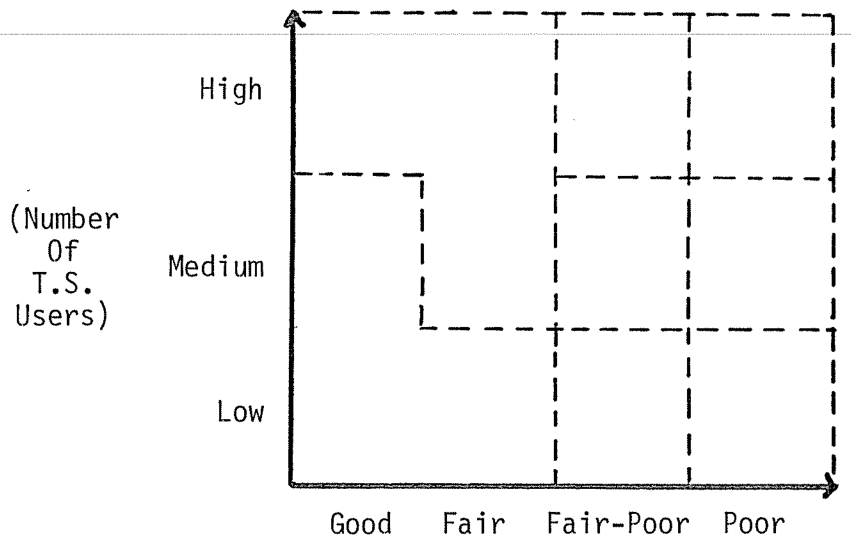


Nine cases.

(Average System Response Time)

(Figure 3.7)

The Restructured Case Space



Eight cases.

(Average System Response Time)

(Figure 3.8)

$\hat{\pi}$ is defined in Step 2.5 as the potential performance improvement associated with a given case space, evaluation function, and M-table. If the case space is restructured, then a new value of $\hat{\pi}$ can be calculated, providing that the necessary raw data has been retained. This is a very significant point, since it means that we can compare alternative case spaces in terms of their associated $\hat{\pi}$ values _without_ further experimentation, to find the best restructuring. True, some additional analysis of the data is required, but it is not difficult.

In practical terms, non-uniform or oddly-shaped cases can be awkward to deal with. The initial division of the L-P space into cases by an n-dimensional grid has the advantage of simplicity, and restructuring approaches which leave intact most of the structure of the original $S/\bar{P}$ are to be preferred. Two simple restructuring approaches with this property are presented below. (Note that each case of the original case space has the form of an n-dimensional rectangular parallelepiped.)

The Analytic Approach: $S/\bar{P}$ may be (recursively) modified by dividing any case in half, by a hyperplane perpendicular to some axis of S/P.

The Synthetic Approach: $S/\bar{P}$ may be (recursively) modified by combining any two adjacent cases.

Figure 3.9 offers a comparison of these two methods of case space restructuring.

Comparison Of Synthetic And Analytic Approaches To Restructuring $S/\bar{P}$

	Synthetic Approach	Analytic Approach
Criteria for Action	Neighboring cases have similar values (or very few observations in a case).	Neighboring cases have dissimilar values (or a case is visited very frequently).
Shape of Cases	May be irregular; not necessarily convex.	Rectangular projection on any planar subspace of S/P.
Mesh (= Fineness of Division Into Cases)	Need finer mesh to start, which means less data per case.	Coarse mesh to start will later be refined only where necessary.
Aid in Understanding S	Gradual steps make trends easier to see; smoother transitions.	Regular regions help; projections are more meaningful.
Ease of Application	May be hard to decide when to merge cases.	If in doubt, splitting is easy to do. It is equivalent to identifying regions where further analysis is needed.
Cost of Implementation	Perhaps somewhat more expensive, since more data must be gathered.	

(Figure 3.9)

One additional comparison is that in the synthetic approach (so-named because of its synthesis of cases, and not because it is artificial in any way), all dimensions of S/P are regarded as equally important. In the analytic approach, a particular dimension may be singled out for splitting, since it is possible that one (or more) of the dimensions is more important than the others in determining the proper control parameter settings. Another way to say this is that the load-performance functions which determine S/P are regarded as passive in the synthetic approach (where cases are joined), and active in the analytic approach (where cases are subdivided).

Case space restructuring can be viewed as an instantaneous process where all changes are made concurrently, but it is intriguing to contemplate the alternative of sequential restructuring. The experimenter may proceed sequentially through the dimensions of S/P, so that restructuring within a given dimension depends on the restructuring done in previously-considered dimensions, and on the order in which the dimensions were considered. The distinction between sequential and concurrent restructuring is independent of the choice of a synthetic or analytic approach to restructuring.

Case space restructuring more drastic than that provided by the two above approaches may be done by returning to Steps 1.5 or 1.6, to reselect the dimensions of S/P, or to repartition them completely.

For convenience, restructured versions of the case space $S/\overline{P}$ will also be referred to as $S/\overline{P}$. This is appropriate, since restructuring usually will leave intact most of the case-structure of $S/\overline{P}$.

Iterative Pathways

It would be agreeable to an experimenter to proceed in a straight line through the steps of the methodology (as in Figure 3.6) to achieve improved system performance. It would also be somewhat surprising to advance in such an unswerving fashion. Case space restructuring, discussed in the previous section, represents one iterative pathway which returns the experimenter to an earlier step of the methodology. In fact, there are compelling reasons to consider other such loops.

At each step of the methodology, the experimenter must make choices or decisions. Sometimes, this is easy to do. More often, there is not really sufficient information or experimental evidence available on which to base selections or crucial decisions. The experimenter will naturally use any knowledge or intuitive understanding he has about system structure and behavior in addition to the data available at the moment of decision. However, it is inevitable that important choices and decisions must be reconsidered after further analysis and experimentation have taken place. If the experimenter knew all the right answers beforehand, there would be no need to experiment! This argues for the inclusion in the methodology of the additional interative pathways which are described below:

a) Return to Step 1.1 to try modifying different control parameters. If it becomes clear at some point that S is relatively insensitive to changes made to C, then other control parameters should be chosen for experimentation. Perhaps system performance is dominated by parameters which have not been experimented on but should be. There is no great merit (or profit) in expending large amounts of time and computer resources to squeeze out one tenth of a percent improvement from the system.

Of course, if modifications to C have been successful, the experimenter may also want to try modifications to other parameters, in hopes of further improving system performance.

b) Return to Step 1.4 to alter the components of G or their relative weights. As we have stated before, G is a policy function. Changing the way G is calculated does not necessarily signal a significant policy change. This kind of action may also be taken if it seems that the components and weights of G do not accurately represent, or implement, the preferences of the experimenter with respect to good system performance.

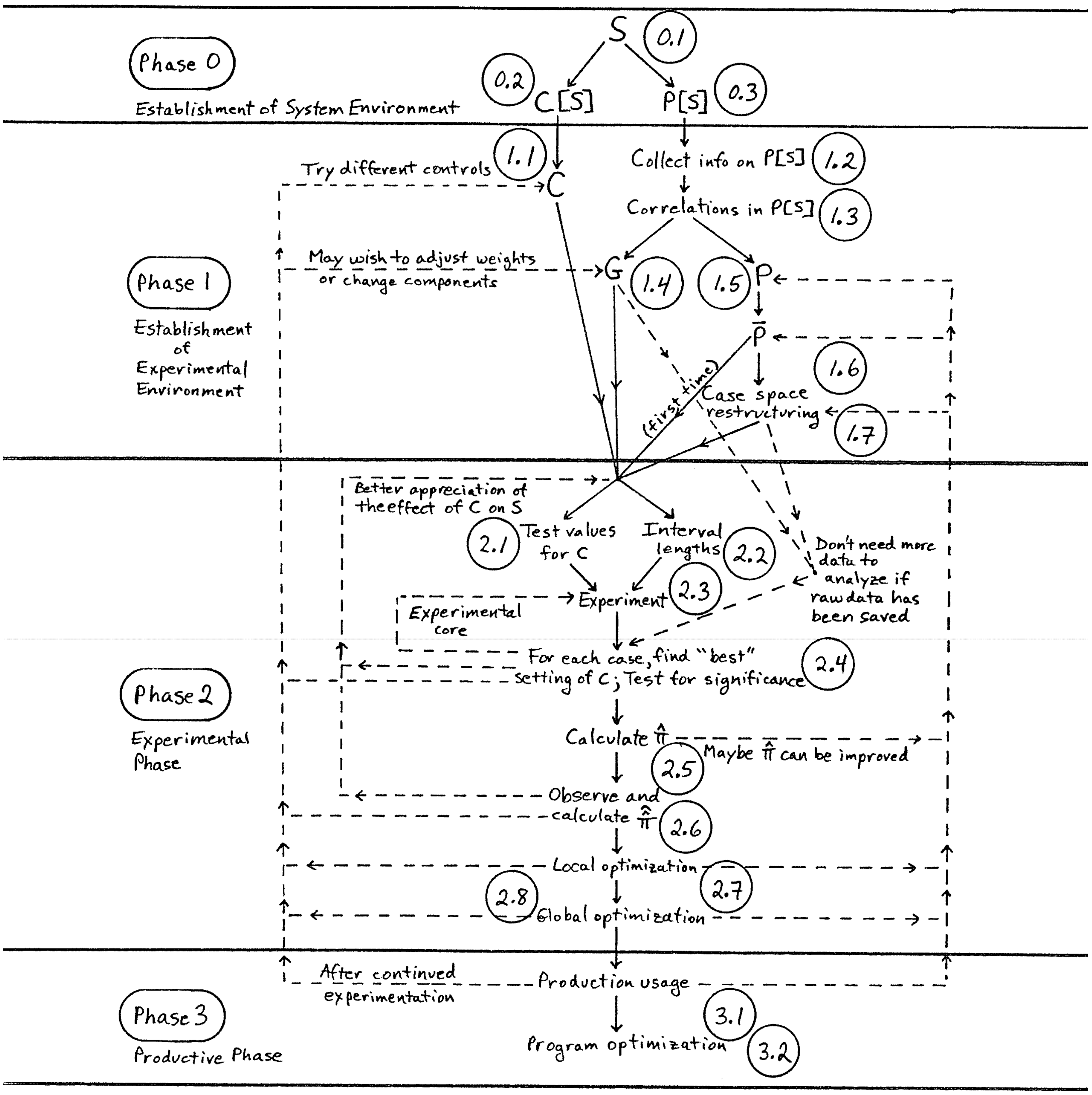
For example, if the components of G include average batch turnaround time, it might be possible to increase G at the expense of the variance of this component. Adding the variance of batch turnaround time as an additional component of G might correct this situation.

c) Return to Steps 2.1 and 2.2 to change control parameter test values, or the lengths of the learning and transient intervals. This is a fairly routine loop to take after the experimenter has a better understanding of the effect of C on S.

d) Return to Step 2.3 (from Step 2.4) to experiment further. This iterative pathway is the experimental core. Experimentation should continue until the differences among control parameter settings have emerged, and can be shown to be statistically significant. If no differences appear, the alternative pathways a), b), and c) above should be considered and compared.

Figure 3.10 shows the iterative pathways (represented by dashed lines) which complete the methodology.

The Methodology With Iterative Pathways



(Figure 3.10)

IV. Summary And Future Research

A system-independent methodology has been presented which focuses on the use of dynamic modification of operating system control parameters to improve system performance, as measured by an evaluation function. The first two phases of the methodology serve to establish the environment for experimentation, and are followed by an experimental phase where alternative parameter settings and modifications are compared by experimenting on the live system. A productive phase then uses the results of experimentation to modify key control parameters on a periodic basis, in order to respond to fluctuations in system load and performance. Throughout the formal presentation of the methodology, concomitant statistical and optimization techniques as well as iterative pathways within the methodology have been emphasized.

We are currently experimenting on the UNIVAC 1100 computer system at the Madison Academic Computing Center of the University of Wisconsin, which serves a large and varied user community. The parameters being modified control the initial core quanta allocated to batch and timesharing runs. The initial core quantum of a program is essentially the length of time it is guaranteed to remain in core before becoming swappable. If the initial quanta are too low, swapping overhead increases; if they are too high, response time for timesharing users suffers. It is our hypothesis that the parameters which control core quantum size should be modified periodically as the size and composition of the workload on the 1110 vary throughout the day. We are using the methodology presented in this report to test that hypothesis, and to gain a better understanding of the practicality of the methodology itself.

V. Bibliography

[BADEL, 1974] Badel, M. *et al*, "Adaptive Optimization of the Performance *of a* Virtual Memory Computer," Rapport de Recherche n^{o} 88, Institut de Recherche d'Informatique et d'Automatique, Rocquencourt, France, November, 1974.

[BARD, 1973] Bard, Y., "Experimental Evaluation of System Performance," *IBM Systems Journal*, Vol. 12, No. 3, 1973, 302-315.

[BARD, 1975] Bard, Y., "An Experimental Approach to System Tuning," IBM Cambridge Scientific Center, Tech Report No. G320-2108, October, 1975.

[BARD & SURYANARAYANA, 1971] Bard, Y., and Suryanarayana, K.V., "On the Structure of CP-67 Overhead," in *Statistical Computer Performance Evaluation* (ed. W. Freiberger), Academic Press, 1972.

[BERNSTEIN & SHARP, 1971] Bernstein, A.J., and Sharp, J.C., "A Policy-Driven Scheduler for a Time-Sharing System," *CACM*, Vol. 14, No. 2, Feb. 1971, 74-78.

[BLEVINS & RAMAMOORTHY, 1971] Blevins, P.R., and Ramamoorthy, C.V., "Aspects of a Dynamically Adaptive Operating System," Tech Report 132, University of Texas, Austin, Texas, July, 1971.

[BOX & DRAPER, 1969] Box, G.E.P., and Draper, N.R., *Evolutionary Operation*, John Wiley and Sons, N.Y., 1969.

[BOX *et al*, 1978] Box, G.E.P, Hunter, W.G., and Hunter, J.S. *Statistics for Experimenters*, John Wiley and Sons, New York, 1978.

[BOX & WILSON, 1951] Box, G.E.P, and Wilson, K.B., "On the Experimental Attainment of Optimum Conditions," *Journal of the Royal Statistical Society of Britain*, Vol. 13, No. 1, 1951.

[BUEHLER *et al*, 1964] Buehler, R.J., Sha, B.V., and Kempthorne, O., "Some Properties of Steepest Ascent and Related Procedures for Finding Optimum Conditions," ONR Tech Report #1, Iowa State University, 1961.

[BUEHLER et al, 1964] Buehler, R.J., Shah, B.V., and Kempthorne, O., "Method of Parallel Tangents," Chem. Eng. Progr., Symp Ser. 60, 1964.

[BUNT, 1975 (1)] Bunt, R.B., "The Effective Treatment of Overload Through A Self-Regulating Scheduler", Tech Report 75-2, Dept. of Computational Science, University of Saskatchewan, Saskatoon, Canada, January 1975.

[BUNT, 1975 (2)] Bunt, R.B., "Self-Regulating Schedulers for Operating Systems", Tech Report No. 76, Dept. of Computer Science, University of Toronto, Canada, January 1975.

[BUNT, 1976] Bunt, R.B., "Adaptive Processor Scheduling Based On Approximating Demand Distribution", Tech Report 76-1, University of Saskatchewan, Saskatoon, Canada, June 1976.

[BUNT, & HUME, 1972] Bunt, R.B., and Hume, J.N.P, "Self-Regulating Operating Systems", INFOR, Vol. 10, No. 3, October 1972.

[CARTON et al, 1973] Carton, E.J., et al, "Some Basic Edge Detection Techniques", Tech Report TR-277, Computer Science Center, University of Maryland, College Park, Maryland, December 1973.

[CHANSON & BISHOP, 1977] Chanson, S.T., and Bishop, C.D., "A Simulation Study of Adaptive Scheduling Policies In Interactive Computer Systems", Performance Evaluation Review, Vol. 6, No. 3, Summer 1977.

[DALKEY, 1969] Dalkey, N.C., Studies in the Quality of Life: Delphi and Decision Making, D.C. Heath & Co., Lexington, Massachusetts, 1969.

[DIXON & MASSEY, 1957] Dixon, W.J., and Massey, F.J. Jr., Introduction to Statistical Analysis, Second Edition, McGraw-Hill, Inc., 1957.

[DOHERTY, 1971] Doherty, W.J., "The Effects of Adaptive Reflective Scheduling", IBM, T.J. Watson Research Center, Tech Report RC 3672, September 1971.

[DRAPER, 1963] Draper, N.R., "Ridge Analysis of Response Surfaces", Technometrics, Vol. 5, No. 4, 1963, 469.

[FERRARI, 1977] Ferrari, D., "An Approach to the Design of a Learning Memory Manager", Proceedings of the 1977 SIGMETRICS/CMG VIII Conference on Computer Performance: Modeling, Measurement, and Management, 1977, 217-224.

[FRANCIS, 1973] Francis, I., "A Comparison of Several Analysis Of Variance Programs", JASA, Vol. 68, No. 344, December 1973.

[FRIEDMAN & WALDBAUM, 1975] Friedman, H.P., and Waldbaum, G., "Evaluating System Changes Under Uncontrolled Workloads; A Case Study", IBM System Journal, Vol. 14, No. 4, 1975, 340-352.

[FU, 1973] Fu, K.S., "Learning Systems", Symposium of AACC, Am. Soc. of Mech. Engineers, 1973.

[GECK, 1979] Geck, A., "Performance Improvement by Feedback Control of the Operating System", 4th Int'l Symp. on Modelling and Performance Evaluation of Computer Systems, Vienna, Austria, February 1979.

[GIBBONS et al, 1977] Gibbons, J.D., Olkin, I., and Sobel, M., Selecting and Ordering Populations: A New Statistical Methodology, John Wiley and Sons, 1975.

[HARTIGAN, 1975] Hartigan, J., Clustering Algorithms, John Wiley and Sons, 1975.

[HAYS, 1973] Hays, W.L., Statistics for the Social Sciences, Second Edition, Holt, Rinehart, and Winston, Inc., 1973.

[KEENEY & RAIFFA, 1976] Keeney, R., and Raiffa, H., Decisions With Multiple Objectives: Preferences and Value Tradeoffs, John Wiley and Sons, 1976.

[KELLY, 1977] Kelly, Capt. J.C., USAF, "Using SIP to Optimize Memory Preference", USE Inc., Spring Conference, Salt Lake City, 1977, pgs. 1-33.

[KEPPEL, 1973] Keppel, G., Design and Analysis: A Researcher's Handbook, Prentice-Hall, Inc., New Jersey, 1973.

[KIMBLETON, 1975] Kimbleton, S.R., "A Heuristic Approach to Computer System Performance Improvement", Tech Report ISI/RR-74-20, Information Sciences Institute, Univ. South California, March 1975.

[KOLENCE, 1973] Kolence, K., "Experiments and Measurements In Computing", SIGME Symposium, Palo Alto, California, February, 1973.

[KRITZINGER et al, 1978] Kritzinger, P.S., Krzesinski, A.E., and Teunissen, P., "Design of a Control System For A Timesharing Computer System", Performance of Computer Installations (ed. D. Ferrari), North-Holland Publishing Co., 1978.

[MAMRAK & DERUYTER, 1977] Mamrak, S.A., and DeRuyter, P., "Statistical Methods for Comparing Computer Services", COMPUTER, Vol. 10, No. 11, November 1977, 32.

[MILLER, 1966] Miller, R.G.Jr., Simultaneous Statistical Inference, McGraw-Hill, 1966.

[MITRANI & HINE, 1977] Mitrani, I., and Hine, J.H., "Complete Parameterized Families of Job Scheduling Strategies", ACTA INFORMATICA 8, Springer-Verlag, 1977, 61-73.

[MYERS, 1976] Myers, R.H., Response Surface Methodology, no publ. given, 1976.

[NIE et al, 1970] Nie, N., et al, SPSS - Statistical Package for The Social Sciences, Second Edition, McGraw-Hill Co., 1970.

[NORTHOUSE & FU, 1973] Northouse, R.A., and Fu, K.S., "Dynamic Scheduling of Large Digital Computer Sysems Using Adaptive Control and Clustering Techniques", IEEE Trans. Systems, Man, and Cybernetics, Vol. SMC-3, No. 3, May 1973, 225-233.

[POTIER et al, 1974] Potier, D., et al, "Adaptive Allocation of CPU Quanta", Rapport de Recherche n° 58, Institut de Recherche d'Informatique et d'Automatique, Rocquencourt, France, April 1974.

[RODRIGUEZ-ROSEL, 1971] Rodriguez-Rosel, J., "Experimental Data On How Program Behavior Affects the Choice of Scheduler Parameters", Proc. ACM Third Symp. on Operating System Principles, Stanford University, 1971.

[RYAN, 1976] RYAN, T.A. Jr., Joiner, B.L., and Ryan, B.F., MINITAB Student Handbook, Duxbury Press, Massachusetts, 1979.

[SAMUEL, 1963] Samuel, A.L., "Some Studies in Machine Learning Using the Game of Checkers", in Computers and Thought (eds E.A. Feigenbaum and J. Feldman), McGraw-Hill, N.Y., 1963, 71-105.

[SAMUEL, 1967] Samuel, A.L., "Some Studies in Machine Learning Using the Game of Checkers II -- Recent Progress", IBM Journal, November 1967, 601-617.

[SHARP, 1973] Sharp, J.C., "An Analysis and Evaluation of Static and Adaptive Policy-Driven Operating System Schedulers", Workshop of ACM SIGME Symp. on Measurement Evaluation, February 1973.

[SHARP & ROBERTS, 1974] Sharp, J.C. and Roberts, J.N., "An Adaptive Policy-Driven Scheduler", Performance Evaluation Review, Vol. 3, No. 4, December 1974, 199.

[SLAGLE, 1971] Slagle, J.R., Artificial Intelligence: The Heuristic Programming Approach, McGraw-Hill, 1971.

[SLAGLE & BURSKY, 1971] Slagle, J.R., and Bursky, P., "Experiments in Automatic Learning for a Multipurpose, Heuristic Program", CACM, Vol. 14, No. 2, 1971, 91-99.

[SMITH, 1973] Smith, M.H., "A Learning Program Which Plays Partnership Dominoes", CACM, Vol. 16, No. 8, August 1973, 462.

[SMITH et al, 1977] Smith, R.G., et al, "A Model For Learning Systems", Heuristic Programming Project Memo 77-14, STAN-CS-77-605, Departments of CS and EE, Stanford University, 1977.

[STEVENS, 1975] Stevens, B.A., "A Note on Figure of Merit", Performance Evaluation Review, Vol. 4, No. 1, 1975, 13.

[TEOREY & PINKERTON, 1972] Teorey, T.J., and Pinkerton, T.B., "A Comparative Analysis of Disk Scheduling Policies", CACM, Vol. 15, No. 3, March 1972.

[WASSAN, 1969] Wassan, M.T., Stochastic Approximation, Cambridge University Press, Cambridge, 1969.

[WILDE, 1964] Wilde, D.J., Optimum Seeking Methods, Prentice-Hall, Inc., 1964.

[WILKES, 1971] Wilkes, M.V., "Automatic Load Adjustment in Timesharing Systems", Proc. ACM Workshop on