The Algorithm Selection Problem*

JOHN R. RICE

Department of Computer Science
Purdue University
West Lafayette, Indiana

1. Introduction

The problem of selecting an effective or good or best algorithm arises in a wide variety of situations. The context of these situations often obscures

* This work was partially supported by the National Science Foundation through Grant GP-32940X. This chapter was presented as the George E. Forsythe Memorial Lecture at the Computer Science Conference, February 19, 1975, Washington, D. C.

the common features of this selection problem, and the primary purpose of this chapter is to formulate abstract models appropriate for considering it. Within the framework established by these models we present a variety of questions that can (and usually should) be asked in any specific application.

It should be made clear that we do not believe that these models will lead directly (by simple specialization) to superior selection procedures. This will always require exploitation of the specific nature of the situation at hand. Even so, we do believe that these models will clarify the consideration of this problem and, in particular, show that some approaches used are based on naive assumptions about the selection process.

Three concrete examples follow, which the reader can use to interpret the abstractions in this chapter.

(a) **Quadrature** One is given a function $f(x)$, an interval $[a,b]$, and a tolerance $\epsilon > 0$. One is to select an algorithm to estimate

$$\int_a^b f(x)\,dx$$

which is efficient [uses few evaluations of $f(x)$] and reliable (produces an estimate within the specified tolerance).

(b) **Operating Systems** One is given an environment for a large computer operation. Information known includes the mix of jobs between batch, interactive and semi-interactive, some basic characteristics of these classes of jobs, and the characteristics of the computer operation. One is to select an algorithm to schedule the execution of these jobs that produces (1) high batch throughput, (2) good response to interactive jobs, (3) good service to semi-interactive jobs, and (4) high priority fidelity.

(c) **Artificial Intelligence** One is given a description of the game Tic-Tac-Toe. One is to select an algorithm to play the game that is effective, i.e., never loses and wins whenever an opponent's mistake allows it.

A selection procedure is invariably obtained by assigning values to parameters in general "form." More precisely, the selection procedure itself is an algorithm and a specific class of algorithms is chosen with free parameters, which are then chosen so as to satisfy (as well as they can) the objectives of the selection problem. Classical forms include polynomials (with coefficients as parameters) and linear mappings (with matrix coefficients or weights as parameters). Other relevant forms are decision trees (with size, shape, and individual decision elements as parameters) and programs (with various program elements as parameters).

The models presented here are primarily aimed at algorithm selection problems with the following three characteristics:

(d) **Problem Space** The set of problems involved is very large and quite diverse. This set is of high dimension in the sense that there are a number of independent characteristics of the problems that are important for the algorithm selection and performance. There is usually considerable uncertainty about these characteristics and their influences.

(e) **Algorithm Space** The set of algorithms that needs to be considered is large and diverse. Ideally there may be millions of algorithms, and practically there may be dozens of them. In counting algorithms we do not distinguish between two that are identical except for the value of some numeric parameter. Again this set is of high dimensions and there is uncertainty about the influence of algorithm characteristics.

(f) **Performance Measure** The criteria to measure the performance of a particular algorithm for a particular problem are complex and hard to compare (e.g., one wants fast execution, high accuracy, and simplicity). Again there is considerable uncertainty in assigning and interpreting these measures.

2. Abstract Models

2.1 The Basic Model and Associated Problems

We describe the basic abstract model by the diagram in Fig. 1. The items in this model are defined below in detail so as to make the nature of the model completely clear:

- $\mathcal{P}$ problem space or collection
- x member of $\mathcal{P}$, problem to be solved
- $\mathcal{A}$ algorithm space or collection

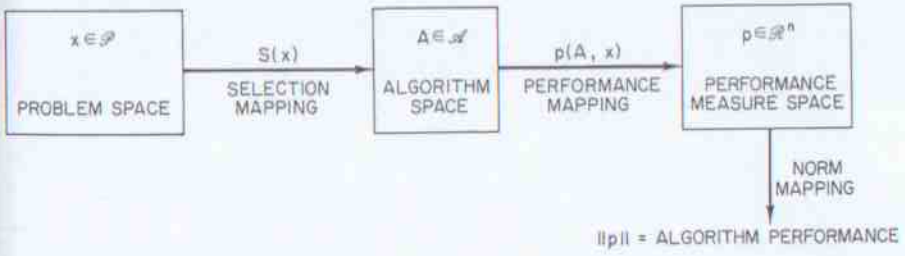


FIG. 1. Schematic diagram of the basic model for the algorithm selection problem. The objective is to determine $S(x)$ so as to have high algorithm performance.

A member of $\mathfrak{A}$, algorithm applicable to problems from $\mathcal{P}$
S mapping from $\mathcal{P}$ to $\mathfrak{A}$
$\mathfrak{R}^n$ n-dimensional real vector space of performance measures
p mapping from $\mathfrak{A} \times \mathcal{P}$ to $\mathfrak{R}^n$ determining performance measures
$\|\ \|$ norm on $\mathfrak{R}^n$ providing one number to evaluate an algorithm's performance on a particular problem

For completeness we now state the:

(a) Algorithm Selection Problem Given all the other items in the above model, determine the selection mapping $S(x)$.

There must, of course, be some criteria for this selection and we present four primary ones:

(A) *Best selection.* Choose that selection mapping $B(x)$ which gives maximum performance for each problem:

$$\|p(B(x),x)\| \geq \|p(A,x)\| \qquad \text{for all} \quad A \in \mathfrak{A}$$

(B) *Best selection for a subclass of problems.* One is to choose just one algorithm to apply to every member of a subclass $\mathcal{P}_0 \subset \mathcal{P}$. Choose that selection mapping $S(x) = A_0$ which minimizes the performance degradation for members of $\mathcal{P}_0$ [compared to choosing $B(x)$]:

$$\max_{x \in \mathcal{P}_0} [\|p(B(x),x)\| - \|p(A_0,x)\|]$$
$$\leq \max_{x \in \mathcal{P}_0} [\|p(B(x),x)\| - \|p(A,x)\|] \qquad \text{for all} \quad A \in \mathfrak{A}$$

(C) *Best selection from a subclass of mappings.* One is to restrict the mapping $S(x)$ to be of a certain form or from a certain subclass $\mathfrak{S}_0$ of all mappings from $\mathcal{P}$ to $\mathfrak{A}$. Choose that selection mapping $S^*(x)$ from $\mathfrak{S}_0$ which minimizes the performance degradation for all members of $\mathcal{P}$:

$$\max_{x \in \mathcal{P}} [\|p(B(x),x)\| - \|p(S^*(x),x)\|]$$
$$\leq \max_{x \in \mathcal{P}} [\|p(B(x),x)\| - \|p(S(x),x)\|] \qquad \text{for all} \quad S \in \mathfrak{S}_0$$

(D) *Best selection from a subclass of mappings and problems.* One is to choose just one algorithm from a subclass $\mathfrak{S}_0$ to apply every member of a subclass $\mathcal{P}_0 \subset \mathcal{P}$. Choose that selection mapping $S^*(x)$ from $\mathfrak{S}_0$ which minimizes the performance degradation for all members of $\mathcal{P}_0$:

$$\max_{x \in \mathcal{P}_0} [\|p(B(x),x)\| - \|p(S^*(x),x)\|]$$
$$\leq \min_{S \in \mathfrak{S}_0} \max_{x \in \mathcal{P}_0} [\|p(B(x),x)\| - \|p(S(x),x)\|]$$

These four criteria do not exhaust the meaningful criteria but they do illustrate the principal ideas. There are five main steps to the analysis and solution of the algorithm selection problem:

Step 1 (Formulation). Determination of the subclasses of problems and mappings to be used.
Step 2 (Existence). Does a best selection mapping exist?
Step 3 (Uniqueness). Is there a unique best selection mapping?
Step 4 (Characterization). What properties characterize the best selection mapping and serve to identify it?
Step 5 (Computation). What methods can be used actually to obtain the best selection mapping?

The reader familiar with the theory of approximation of functions will recognize this framework, within which we may put that classical theory. The space $\mathcal{P}$ is a function space and the algorithm space $\mathfrak{A}$ may be identified with a subspace of $\mathcal{P}$. The algorithm enters as the means of evaluating elements of $\mathfrak{A}$. The performance mapping is

$$p(A,x) = \|x(t) - A(t)\|_{\mathcal{P}}$$

where the norm is taken on $\mathcal{P}$. Thus the performance measure space is $\mathfrak{R}^1$ and the norm mapping is trivial.

There are two remarks needed about this observation. First, the body of significant material in approximation theory is large. It would require, no doubt, from 2000 to 4000 pages to present a reasonably complete and concise exposition of the results currently known. This implies that there is a very rich body of material waiting to be applied to the algorithm selection problem, either directly or by analogy. Second, and more important, the algorithm selection problem is an essential extension and generalization of approximation theory. We will see concrete examples of this problem where the current theory of approximation has nothing relevant to apply except by the faintest analogies.

Two concrete examples of the model are discussed in detail in Sections 3 and 4 of this chapter. We present a third, simpler one from the area of artificial intelligence.

(b) Example: A Game-Playing Problem We are to devise an algorithm for playing Tic-Tac-Toe. The problem space is the set of partial games of Tic-Tac-Toe. While this number is large, there are in fact only 28 distinct reasonable games if one eliminates blunders, symmetries, and board rotations. The space $\mathfrak{A}$ may be represented as a space of large tables of responses for each situation. However, we restrict our selection to a decision tree that involves only the existence of immediate winning positions and

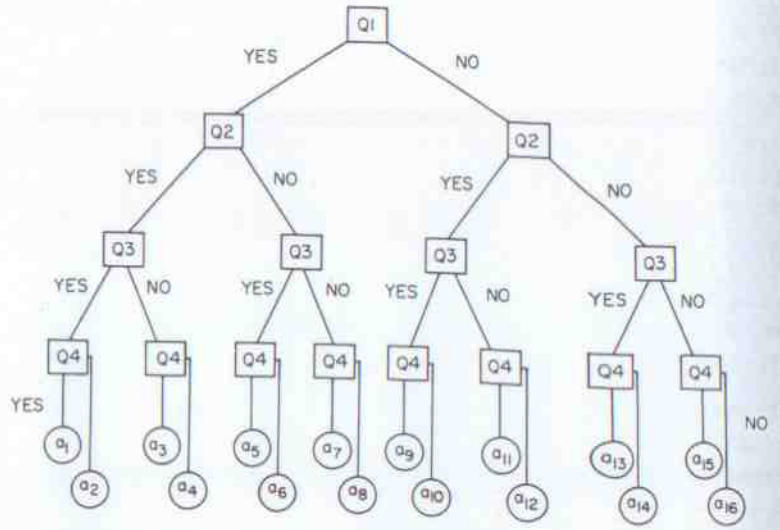


FIG. 2. The form of the selection mapping for the Tic-Tac-Toe example. Each a_i is one of five moves. Q1: Do I have a winning position? Q2: Does opponent have a winning position? Q3: Is the center free? Q4: Is a corner free?

vacant position types. The algorithm form may then be represented as shown in Fig. 2. There are 16 parameters a_i, which take on one of the following five values:

(1) Play the winning move.
(2) Block the opponent's win.
(3) Play in the center square.
(4) Play in a corner (first free one clockwise from upper right).
(5) Play in a side (first free one clockwise from right).

This example is so simple that one can make immediate assignments of certain of the values of the a_i. Experiments have shown that a variety of crude schemes for computing values of the a_i (selecting the best algorithm) work very quickly. Nevertheless, it is still of interest to reflect upon how one would compute this if one had no *a priori* information about the game.

2.2 The Model with Selection Based on Features

An examination of various instances of the algorithm selection problem shows that there is another ingredient almost always present. It is sometimes explicit and sometimes not and we call this selection based on features of the problem. This model is described by the diagram in Fig. 3.

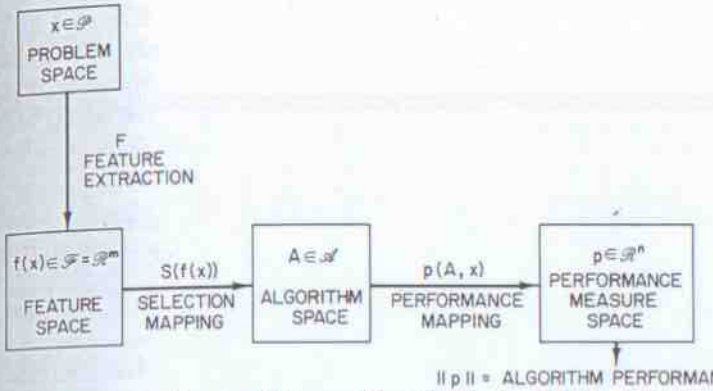


FIG. 3. Schematic diagram of the model with selection based on features of the problem. The selection mapping depends only on the features $f(x)$, yet the performance mapped still depends on the problem x.

The additional definitions for this model are $\mathcal{F}$, feature space identified with $\mathcal{R}^m$ here to suggest it is simpler and of lower dimension than $\mathcal{P}$, and F, mapping from $\mathcal{P}$ to $\mathcal{F}$ that associates features with problems. Note that the selection mapping now depends only on the features $f(x)$, yet the performance mapping still depends on the problem x. The introduction of features may be viewed as a way to systematize the introduction of problem subclasses in the basic model.

The previous statement of the algorithm selection problem and the criteria for selection are still valid for this new model as well as the five steps in the analysis and solution of the problem. The determination of the features to be used is frequently part of the selection process, often one of the most important parts. One may view the features as an attempt to introduce an approximate coordinate system in $\mathcal{P}$. Ideally, those problems with the same features would have the same performance for any algorithm being considered. Since this ideal is rarely achieved, we may pose several specific questions about the determination of features.

(E) *Best features for a particular algorithm.* Given an algorithm A and the dimension m of $\mathcal{F}$, what m features are the best for the prediction of the performance of A? Let $\mathcal{E}(f)$ denote the equivalence class of all those problems $x,y \in \mathcal{P}$ so that $F(x) = F(y) = f$. We then wish to determine the mapping F^* and associated equivalence classes $\mathcal{E}^*(f)$ so that

$$d_m^*(A) = \max_{f \in \mathcal{F}} \max_{x,y \in \mathcal{E}^*(f)} \| p(A,x) - p(A,y) \|$$
$$\leq \max_{f \in \mathcal{F}} \max_{x,y \in \mathcal{E}(f)} \| p(A,x) - p(A,y) \|$$

The selection of best features corresponds to the selection of best approximating subspaces in approximation theory and leads one to ideas of n-widths and entropy of the problem space $\mathcal{P}$. Roughly speaking, if d_m^* is large, then the effective dimension of $\mathcal{P}$ (for the problem at hand) is probably much larger than m and, conversely, if d_m^* is small, then the effective dimension of $\mathcal{P}$ is close to m.

(F) *Best features for a class of algorithms.* Given a set $\mathcal{A}_0 \subset \mathcal{A}$ and the dimension m of $\mathcal{F}$, what m features are the best for prediction of the performance of algorithm $A \in \mathcal{A}_0$? With the previous notation we wish to determine F^* and $\mathcal{E}^*(f)$ so that

$$d_m^*(\mathcal{A}_0) = \max_{f\in\mathcal{F}} \max_{A\in\mathcal{A}_0} \max_{x,y\in\mathcal{E}^*(f)} \| p(A,x) - p(A,y)\|$$

$$\leq \max_{f\in\mathcal{F}} \max_{A\in\mathcal{A}_0} \max_{x,y\in\mathcal{E}(f)} \| p(A,x) - p(A,y)\|$$

(G) *Best features for a subclass of selection mappings.* Given a subclass $\mathcal{S}_0$ of selection mappings from $\mathcal{F}$ to $\mathcal{A}$, what m features are the best for prediction of the performance of algorithms? With the previous notation we wish to determine F^* and $\mathcal{E}^*(f)$ so that

$$d_m^*(\mathcal{S}_0) = \max_{f\in\mathcal{F}} \max_{S\in\mathcal{S}_0} \max_{x,y\in\mathcal{E}^*(f)} \| p(S(f),x) - p(S(f),y)\|$$

$$\leq \max_{f\in\mathcal{F}} \max_{S\in\mathcal{S}_0} \max_{x,y\in\mathcal{E}(f)} \| p(S(f),x) - p(S(f),y)\|$$

The determination of the best (or even good) features is one of the most important, yet nebulous, aspects of the algorithm selection problem. Many problem spaces $\mathcal{P}$ are known only in vague terms and hence an experimental approach is often used to evaluate the performance of algorithms over $\mathcal{P}$. That is, one chooses a sample from $\mathcal{P}$ and restricts considerations to this sample. An appropriate sample is obviously crucial to this approach and if one has a good set of features for $\mathcal{P}$, then one can at least force the sample to be representative with respect to these features. Note that the definition of best features is such that they are the items of information most relevant to the performance of algorithms for the problem at hand.

In some well-understood areas of computation there is a generally agreed upon (if not explicitly stated) set of features. For example, consider the problem of solving a linear system $Ax = b$ of equations. The features include descriptors like small order, sparse, band, diagonally dominant, positive definite, and ill conditioned. Given values for these features, an experienced numerical analyst can select an appropriate algorithm for this problem with considerable confidence. The selection problem for quadrature is already much more difficult and the solution of simultaneous systems of nonlinear equations is very poorly understood. If this situation exists for problems that have been studied for one or two centuries then one should not be surprised by the difficulties and uncertainties for problems that have just appeared in the past one or two decades.

2.3 Alternate Definitions of Best for the Models

In the preceding sections we have uniformly taken a minimax approach to the definition of best or optimum selection. That is, we have minimized the effect of the worst case. It is reasonable to ignore the performance for the worst case and instead consider optimizing some sort of average behavior. In this section we exhibit the resulting mathematical problems corresponding to using a least squares or least deviation approach (these correspond to L_2 and L_1 optimization in mathematical terms). We have identified seven problems labeled A through G. Problem A is unaffected by these considerations so let us consider problem B (best selection for a subclass of problems). We use the notation introduced with the original mathematical statement of this problem, which is:

Minimax approach

$$\max_{x\in\mathcal{P}_0} [\| p(B(x),x)\| - \| p(A^*,x)\|]$$

$$\leq \max_{x\in\mathcal{P}_0} [\| p(B(x),x)\| - \| p(A,x)\|] \quad \text{for all} \quad A \in \mathcal{A}$$

The corresponding mathematical statements for the least squares and least deviation approach are

Least squares approach

$$\int_{\mathcal{P}_0} [\| p(B(x),x)\| - \| p(A^*,x)\|]^2\, dx$$

$$\leq \int_{\mathcal{P}_0} [\| p(B(x),x)\| - \| p(A,x)\|]^2\, dx \quad \text{for all} \quad A \in \mathcal{A}$$

Least deviations approach

$$\int_{\mathcal{P}_0} |\, \| p(B(x),x)\| - \| p(A^*,x)\|\,|\, dx$$

$$\leq \int_{\mathcal{P}_0} |\, \| p(B(x),x)\| - \| p(A,x)\|\,|\, dx \quad \text{for all} \quad A \in \mathcal{A}$$

The use of integrals in these formulations implies that a topology has been

introduced in the problem space $\mathcal{P}$. Many common examples for $\mathcal{P}$ are discrete in nature and in these cases the topology introduced reduces the integrals to sums. This technicality is unlikely to cause real difficulties and we continue to use integrals, as this gives the neatest formulations. Note that the only difference between the two new formulations is the exponent (2 or 1) in the integrand. Thus we may avoid repeating these formulations twice by making this a variable, say r, which has values 1 or 2. Note that in approximation theory it is shown that minimax is the limiting case as $r \to \infty$ so that all three approaches can be expressed in one formulation with r as a parameter.

We recall that problem C is the best selection from a subclass of mappings. The alternative mathematical formulation of this problem is

$$\int_{\mathcal{P}} |\, ||\, p(B(x),x)|| - ||\, p(S_0(x),x)||\, |^r\, dx$$

$$\leq \int_{\mathcal{P}} |\, ||\, p(B(x),x)|| - ||\, p(S(x),x)||\, |^r\, dx \qquad \text{for all} \quad S \in \mathcal{S}_0$$

The alternative formulation for problem D is identical to this except that the problem subclass $\mathcal{P}_0$ replaces $\mathcal{P}$ as the domain of integration.

The next three problems involve features and we choose to use a consistent approach for the reformulations. That is, if we use least squares on the problem space we also use it on the feature space $\mathcal{F}$ and the algorithm space $\mathcal{A}$. If we set

$$d_m{}^r(A,\mathcal{E}) = \int_{f\in\mathcal{F}} \left[\int_{x,y\in\mathcal{E}(f)} \int ||\, p(A,x) - p(A,y)||^r \right]^{1/r}$$

then for problem E (best feature for a particular algorithm), the objective is to find the feature mapping F^* and associated equivalence classes $\mathcal{E}^*(f)$ that minimize $d_m{}^r(A,\mathcal{E})$, i.e.,

$$d_m{}^r(A) = d_m{}^r(A,\mathcal{E}^*) = \min_{\mathcal{E}} d_m{}^r(A,\mathcal{E})$$

For problem F we introduce

$$d_m{}^r(\mathcal{A}_0,\mathcal{E}) = \int_{f\in\mathcal{F}} \left[\int_{A\in\mathcal{A}_0} \int_{x,y\in\mathcal{E}(f)} \int ||\, p(A,x) - p(A,y)||^r \right]^{1/r}$$

and then the objective is to determine F^* and associated $\mathcal{E}^*(f)$ so that

$$d_m{}^r(\mathcal{A}_0) = d_m{}^r(\mathcal{A}_0,\mathcal{E}^*) = \min_{\mathcal{E}} d_m{}^r(\mathcal{A}_0,\mathcal{E})$$

A similar approach to problem G yields a similar expression except that the integral over $\mathcal{A}_0$ is replaced by an integral over $\mathcal{S}_0$.

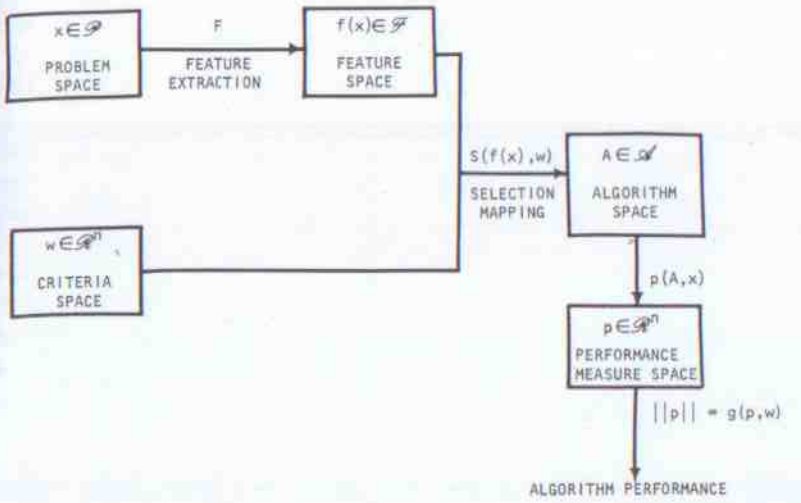


FIG. 4. Schematic diagram of the model with selection based on problem features and variable performance criteria.

In many practical problems there is little to guide one in the choice of a particular formulation of the mathematical optimization problem, i.e., should we choose $r = 1$, 2, or ∞? These choices might not be particularly significant in the larger context, but they are very significant in determining the difficulty of the resulting mathematical optimization problem. A lesson learned from practical approximation theory might be applicable in this larger context. *This lesson is, roughly, that the crucial ingredients for success are proper choices of the subclasses* $\mathcal{P}_0$, $\mathcal{A}_0$, *and* $\mathcal{S}_0$. Once these are made properly then the mathematical optimization should be made for that value of r which gives the least difficulty. If the problem is completely linear then $r = 2$ (least squares) almost always results in the least mathematical difficulty. The situation is more variable for nonlinear problems. Note that there are practical approximation problems where the choice of r is crucial and no doubt there are similar cases for the algorithm selection problem. We are saying that the choice of r is important only in a small number of instances.

2.4 The Model with Variable Performance Criteria

We have assumed so far that there is a fixed way to measure the performance of a particular algorithm for a particular problem. There are, however, many situations where it is reasonable to view the performance criteria as input to the selection problem. Consider, for example, the selec-

tion of a program to solve ordinary differential equations and the criteria of speed, accuracy, reliability, and ease of use. In different situations the weight given to each of these might vary from almost zero to almost 100%. A model for this version of the selection problem is shown in Fig. 4.

The additional definition for this model is g, norm function from $\mathfrak{R}^n \times \mathfrak{R}^n$ to R^1, which measures the algorithm performance $p(A,x)$ with the criterion w. Some of the mappings have now changed domains, but their nature is the same. The choice of $\mathfrak{R}^n$ for the criteria space is clearly arbitrary (and perhaps unnecessarily restrictive) but it is natural for the most common choice of the norm function $g(p,w) = p \cdot w$.

We can at this point formulate new versions of the algorithm selection problem involving the criteria space. The variables of these formulations are

Problem subclasses:	$\mathcal{P}_0$
Algorithm subclasses:	$\mathcal{A}_0$
Selection mapping subclasses:	$\mathcal{S}_0$
Feature space:	$\mathcal{F}$
Norm mapping:	g

The number of interesting combinations is now quite large and we refrain from formulating all of them. Some of the more important problems are as follows.

(H) *Best selection for a given criterion.* We assume that $g(p,w)$ is known, that $\mathcal{F} = \mathcal{P}$ (and F is the identity), and that w is given. The problem then is to determine that selection mapping $B(x,w)$ which gives maximum performance:

$$g(p(B(x,w),x)w) \geq g(p(A,x),w) \qquad \text{for all} \quad A \in \mathcal{A}$$

(I) *Best selection from a subclass of mappings for a given criterion and feature space.* We restrict S to a subclass $\mathcal{S}_0$ of all mappings from $\mathcal{F} \times \mathfrak{R}^n$ to A and, for a particular specified value of w and problem x, we wish to determine the best mapping $S^*(x,w)$ so that

$$g(p(S^*(f(x),w),x),w) \geq g(p(S(f(x),w),x),w) \qquad \text{for all} \quad S \in \mathcal{S}_0$$

(J) *Best selection from a subclass of mappings, problems, and algorithms for a given criterion and feature space.* This is a model of perhaps the most realistic situation. We have the feature space $\mathcal{F}$ and norm function g specified. We restrict ourselves to subclasses $\mathcal{S}_0$, $\mathcal{P}_0$, and $\mathcal{A}_0$ of selection mappings, problems, and algorithms, respectively. Note we have $\mathcal{S}_0$: $\mathcal{F} \times \mathfrak{R}^n \rightarrow$

$\mathcal{A}_0$. Within this framework we wish to select that mapping S^* so that

$$\max_{w \in \mathfrak{R}^n} \max_{x \in \mathcal{P}_0} g(p(B(x,w),x),w) - g(p(S^*(f(x),w),x),w)$$
$$\leq \max_{w \in \mathfrak{R}^n} \max_{x \in \mathcal{P}_0} g(p(B(x,w),x),w) - g(p(S(f(x),w),x),w)$$

for all $S \in \mathcal{S}_0$. Note that $g(p(B(x,w),x),w)$ is the best possible performance and the other g terms are the performances of the algorithms actually selected.

We note that the abstract model presented in this section could be elaborated upon considerably. The study of the theoretical questions of the existence, uniqueness, and characterization of best selection mappings and features can be expanded to fill a thick monograph. Those familiar with the mathematician's ability to develop theoretical structures from simple models can visualize how this would be done. However, the crucial point of a model is not its theoretical structure but its relevance to underlying real world problems. In other words, does this model allow us to develop better insight, understanding, and analysis of real algorithm selection problems? This question is addressed in the next two sections.

3. Concrete Application—The Selection of Quadrature Algorithms

3.1 The Components in the Abstract Model

The next two sections are independent of one another and each has the following format:

Formulation of the general problem and definition of the relevant spaces,
Examination of concrete cases,
Formulation of a specific and simpler selection problem,
Discussion of the simpler problem and the computations required to solve it.

The general case of the quadrature algorithm selection problem may be expressed in one of the two following ways:

(A) Given a collection of functions (with reasonably well-known attributes), which one of the 15–25 well-known quadrature algorithms should be selected so as to give the best performance?

(B) Given that a program library for a computing center should contain a small (1–4) number of quadrature algorithms, which ones should be selected?

A thorough analysis of these two questions is a formidable task. We will formulate this problem (version B) more precisely and summarize the rather extensive amount of information bearing on the question. Then we formulate a somewhat simpler and more concrete problem and discuss its solution. This general problem is modeled as in Section 2.4, which involves spaces for the problems, the features, the criteria, the algorithms, and the performance measures. These spaces are described as follows:

(a) **Problem Space** This space consists of a rather broad class of functions of one variable. While the population characteristics are not well known, it is likely that the bulk of the functions are simple, smooth, and well behaved, and yet a small but still significant proportion of the functions have properties that cause real difficulty in quadrature. The possible properties are illustrated by the feature space.

(b) **Feature Space** The features of these problems that should be included are indicated by a key word followed by a short explanation:

Smoothness: either mathematical or intuitive.
Jumps: jump discontinuities of various sizes are present (or absent).
Singularities: local behavior of the form t^α, $-1 < \alpha < 1$ or $\alpha > 1$ and not integer; $\log t$, etc.
Peaks: small subintervals where the function makes a radical change in size; may be actual peaks or "smoothed" jump discontinuities.
Oscillations: oscillatory behavior of various amplitudes, frequencies, and extent.
Round-off: the presence of significant random uncertainty in the value of the function.
Symbolic: some attributes may be obtained by a cursory examination of the function's description.
Accuracy: the desired accuracy of the quadrature estimate.
Domain: the interval of integration (might be infinite).

No doubt there are other significant problem features that have been overlooked in this list.

(c) **Algorithm Space** There are about 15 or 20 quadrature algorithms that have been completely defined and studied to a certain extent in the literature. In addition, there are a number of very classical algorithms (e.g., Simpson's rule) that must be considered even though they are not standardized (i.e., they really are classes of algorithms). Note that this small number is from a *very large population* of many millions (see Rice, 1975). The actual algorithms one might consider are mentioned in the various references, and many of them are named later.

(d) **Performance Measures** The most commonly considered measures of performance are *work* (measured in number of function evaluations) and *reliability* (1 if the requested accuracy is achieved). Other important algorithm characteristics are *ease of use*, *understandability* (for possible modification), *memory* requirements (both for the algorithm and problem data generated), and ease of *analysis*.

(e) **Criteria Space** This consists of some numbers designed to weight the relative importance of the performance measures. The measures in this case are not very compatible and it is difficult to find a completely satisfactory method of comparing the various measures. Scaling all the measures from zero to one and then applying simple weights is a naive approach with considerable appeal. Comparisons that involve step functions are more realistic but less tractable to use or describe to users.

3.2 Review of Previous Work on Quadrature Algorithm Evaluation

A substantial number of experimental tests have been made and reported in the literature. The functions involved have primarily been chosen from one of the following three *test function sets* (samples from the problem space):

(A) Casaletto *et al.* (1969): a set of 50 functions,
(B) Kahaner (1971): a set of 21 functions,
(C) de Boor (1971): three performance profiles.

There is a small overlap among these sets and some authors have used various subsets, occasionally with a few additions.

There have been ten substantial testing efforts reported, which are listed below in chronological order. We indicate the test functions used (by A, B, or C), the requested accuracies (by ϵ values), and the algorithms involved. The algorithms are named and described, but for detailed references one must refer to the test reports.

(1) Casaletto *et al.* (1969). Complete details not reported. Test set *A* with $\epsilon = 10^{-1}, 10^{-2}, \ldots, 10^{-8}$. Algorithms: QUAD (adaptive Simpson rule), QUADS4 (adaptive 4-point Gauss quadrature), QUADS6 (adaptive 6-point Gauss quadrature), SIMP (adaptive Simpson rule; almost identical with SIMPSN), SIMPSN (adaptive Simpson rule), SQUANK (improved version of SIMPSN), ROMBRG (adaptive Romberg integration), RIEMAN (adaptive Riemann sums).

(2) Kahaner (1971). Extensive tables of detailed results. Test set *B* with $\epsilon = 10^{-3}, 10^{-6}, 10^{-9}$. Algorithms: SIMPSN, SQUANK, GAUSS (adaptive Gauss using 5- and 7-point rules), G96 (96-point Gauss rule),

HAVIE (improved version of ROMB), QABS (combination Romberg and Curtis–Clenshaw), QNC7 (adaptive 7-point Newton–Cotes rule), QUAD (adaptive 10-point Newton–Cotes rule), RBUN (adaptive Romberg), ROMB (standard Romberg), SHNK (Romberg type using Wynn's ϵ-algorithm for extrapolation).

(3) de Boor (1971). Results compatible with Kahaner plus graphs. Test set B with $\epsilon = 10^{-3}, 10^{-6}, 10^{-9}$ plus test set C. Algorithm: CADRE (adaptive Romberg with cautious extrapolation).

(4) Gentleman (1972). Considerable detail. Test set A with $\epsilon = 10^{-1}, 10^{-2}, \ldots, 10^{-8}$. Algorithm: CCQUAD (Curtis–Clenshaw quadrature).

(5) Patterson (1973). Partial results reported involving CADRE and QSUBA. Test set selected from A and B plus three others; total of 13 functions. Algorithms: CADRE, SQUANK, QSUB (iterated Gauss–Kronrod rules up to 255 points), QSUBA (adaptive version of QSUB).

(6) Piessens (1973a). Complete details not reported. Test set A with $\epsilon = 10^{-2}, 10^{-3}, \ldots, 10^{-13}$. Algorithms: CCQUAD, SQUANK, AIND (adaptive Gauss–Kronrod rules up to 21 points), HRVINT (improved version of HAVIE, (adaptive Romberg).

(7) Piessens (1973b). Considerable detail given, some round-off effects studied. Test set A with $\epsilon = 10^{-5}, 10^{-7}$ (with noise), 0. Algorithms: AIND, CADRE, SQUANK.

(8) Einarsson (1974). Complete detail for selected cases only. Test set A with $\epsilon = 10^{-1}, 10^{-2}, \ldots, 10^{-6}$. Algorithms: CCQUAD, DRMBIU (IMSL version of Romberg quadrature; two versions), QATR (IBM-SSP version of Romberg quadrature), QAR (IBM-SL-MATH version of Romberg quadrature), ROMINT (Romberg quadrature; algorithm 351 of *Comm. ACM*).

(9) Blue (1975). Considerable detail for a large number of cases plus numerous performance profiles for his own algorithm. Test set B with $\epsilon = 10^{-3}, 10^{-6}, 10^{-9}$. Algorithms: CADRE, QABS, QNC7, QSUBA, QUAD, RBUN, ROMB, SIMPSN, SQUANK, DQUAD (adaptive Romberg with cautious extrapolation).

(10) Krogh and Snyder (1975). Extensive tables of detailed results. Test set of combined nature and several hitherto unused integrands, $\epsilon = 10, 1, \ldots, 10^{-7}$. Algorithms: AIND, CADRE, QNC7, QSUBA, RBUN, GAUS8 (adaptive 8-point Gauss algorithm), SINT (extensive revision of QSUBA).

Also see Lyness and Kaganove (1976) for further discussion on the nature of this problem. This testing has provided much useful information and served to identify some poor algorithms. However, it has not been well enough organized to allow definitive conclusions and there is still considerable doubt about the relative merits of the better algorithms. We note that a much better experiment can be performed.

3.3 A Systematic Evaluation and Selection Approach

We assume the quadrature problem is $\int_0^1 h(t)\,dt$. We choose a feature space with four dimensions:

Feature name	Values assumed	Remarks
Smoothness	0, ½, 1	0 is smooth, 1 is not
Singularity	[−1, 2]	Value is exponent of singularity
Peak	[0, 100]	Strength $= \dfrac{[\text{average size of } h(t) - \text{ave. peak}]}{(\text{peak base})^* (\text{ave. size of peak})}$
Oscillation	[0, 100]	Maximum frequency of oscillation

We choose four one-parameter families of functions that represent each of the features (the performance profiles) and then each coordinate axis of $\mathfrak{F}$ is discretized and families introduced with characteristics of each of the remaining features. Such families can be easily constructed by addition or multiplication (e.g., $|\ t^2-0.25\ |^\alpha$ has a singularity, $\sin[N(t^2+1)]$ is oscillatory, and both $|\ t^2-0.25\ |^\alpha + \sin[N(t^2+1)]$ and $|\ t^2-0.25\ |^\alpha \sin[N(t^2+1)]$ are oscillatory with a singularity). This process gives a test set that produces a grid over the entire feature space. This test set can be combined with accuracy values of $\epsilon = 10^{-2}, 10^{-4}, 10^{-8}, 10^{-12}$, to permit a much more precise measurement of algorithm performance.

There are about a dozen existing algorithms that merit inclusion in this experiment and a little estimation shows that a rather substantial computation is required for this experiment. An important result of the systematic nature of this approach is that one can consider probability distributions in the problem space that induce a probability distribution on the feature space and algorithm performances can be compared (over this problem subdomain) without repeating the experiment.

This suggested experiment is far from the most general of interest and is clearly biased against certain well-known algorithms. For example, SQUANK takes considerable care in handling round-off effects (a feature omitted here) and explicitly ignores oscillations (a feature included here). Thus one would not expect SQUANK to compare favorably with some other algorithms on the basis of this experiment.

We consider two criteria of performance: *efficiency* and *reliability*. These two variables are scaled to the interval [0,1] as follows:

(a) Efficiency Let N_x be the minimum number of integrand evaluations required to solve problem x (this must be estimated for each problem) and

N_A the actual number used by a particular algorithm A. Then the value of the efficiency is $N_x/N_A = p_1(A,x)$ for algorithm A and problem x.

(b) **Reliability** Let ϵ_x be the requested accuracy and ϵ_A the accuracy actually achieved. The value of reliability is then taken to be

$$p_2(A,x) = \begin{cases} 1 - (1 - \epsilon_x/\epsilon_A)^2 & \epsilon_A > \epsilon_x \\ 1/(1 + 0.1(\log \epsilon_A/\epsilon_x)^2) & \epsilon_A < \epsilon_x \end{cases}$$

This places a severe penalty on failing to achieve ϵ_x, and a mild penalty on achieving much more accuracy than ϵ_x. These conventions allow us to find the performance vector $(p_1(A,x),p_2(A,x))$. We introduce a criteria unit vector (w_1,w_2) and the norm of $p(A,x)$ is then

$$|| p(A,x)|| = w_1p_1(A,x) + w_2p_2(A,x)$$

4. Concrete Application—The Selection of Operating System Schedulers

4.1 The Components in the Abstract Model

The general case of this problem may be expressed as follows: Consider a computing installation with a fixed configuration and a work load with reasonably well-known attributes. How should jobs be scheduled in order to give the best service?

A thorough analysis of this problem requires many hundreds of pages and is beyond the scope of this chapter. We will formulate this problem more precisely within the framework provided by the abstract models. This formulation is oriented toward the specific case of the operation in the Purdue University Computing Center, which is a typical example of large-scale, complex operation. Then we describe a simplified version of the current scheduling algorithm and, in turn, formulate a much more specific algorithm selection problem. A discussion is then given of how one could attempt to solve this problem in terms of the information that is known or obtainable.

The abstract model involves spaces for the problems, the features, the criteria, the algorithms, and the performance measures. These spaces are described as follows:

(a) **Problem Space** This space consists of configurations of computer runs, which are mixtures of batch, remote batch, time-shared, and interactive jobs. These configurations are very dynamic in nature and normally only general average values are known for the population characteristics (and most of these values are not known accurately). In addition to very

rapid and substantial changes in the problem characteristics, there are often well-identified long-term variations in the average values of the problem characteristics.

(b) **Feature Space** The features of a configuration of computer runs are a combination of the features of the individual jobs. The features of individual jobs that should be considered are indicated by a keyword plus a short explanation:

Priority: value given by user and computing center.
CPU time: value estimated for job by user.
Memory: value estimated for job by user and observed by operating system; both core and auxiliary memory values may be considered.
I/O requirements: values estimated by user for use of standard devices (printers, punches, disk channels, etc.).
Special facilities: indications of use of less common facilities (e.g., tape units, plotters, graphics consoles).
Program locality and stability: indication of the likelihood of page requests or job roll-outs.

In addition, features of the total problem configuration should be considered as follows:

Batch load: length of the input queue plus average values for some of the job features.
On-line load: number of terminal users plus average values features for the stream of jobs they create.
Interactive load: number of users and nature of system being used.
I/O load: length of queues at the various I/O devices.

No doubt there are other significant problem features not included in this list.

(c) **Algorithm Space** A fair variety of scheduling algorithms have been proposed and analyzed to a certain extent (Coffman and Denning, 1974; Wilkes, 1973). An essential characteristic of successful algorithms is that they are fast to execute (otherwise the system devotes an excessive amount of its resources to scheduling instead of production). This favors some very simple schemes (e.g., round-robin, first-come first-served, simple priority) but one must realize that rather complex algorithms can be fast to execute.

(d) **Performance Measures** The performance of an operating system depends on one's viewpoint—each user wants instant service and the computing center director wants zero angry or dissatisfied customers. Neither of these desires is very realistic, but efforts to measure the progress made

toward satisfying them usually involve throughput and response time. These measures are applied to different classes of jobs as follows:

Batch: small job response; median and maximum turnaround for jobs with small resource requirements.

Batch: large job response; median and maximum turnaround for all batch jobs other than small ones (or special runs).

On-line response: median and maximum response time for common service functions (e.g., fetching a file, editing a line, submitting a batch job).

Interactive response: median and maximum response times for standard short requests.

Throughput: total number of jobs processed per unit time, number of CPU hours billed per day, etc.

(e) Criteria Space This consists of numbers to weight the relative importance of the performance measures. Values of some of these measures can be improved only by making others worse and it is difficult to compare them. Scaling the measures to a standard interval (say 0 to 1) and then applying weights (which sum to unity) is simple, but often satisfactory.

4.2 An Actual Scheduling Algorithm

We present a version of the scheduling algorithm used on the CDC 6500 system at Purdue University (see Abell, 1973). This algorithm has been simplified by omitting features for preventing deadlock, "first pass" priority given initially to all jobs, and job origin priority. Jobs are scheduled according to priority, i.e., if a waiting job has queue priority QP_1 larger than an executing job with queue priority QP_2 and if the central memory CM_2 used by the executing job is large enough for the waiting job (which requires CM_1 in memory) then the executing job is terminated and rolled out and the waiting job is rolled in and placed into execution. In summary, if $QP_1 > QP_2$ and $CM_1 \leq CM_2$ than job 2 is rolled out and replaced by job 1.

The queue priority QP is a function of six priority parameters $\mathbf{r} = (r_1, r_2, r_3, r_4, r_5, r_6)$ as follows:

- r_1 job card priority parameter
- r_2 central memory (current requirement)
- r_3 time remaining on CPU time estimate
- r_4 I/O units remaining on I/O transfer unit estimate
- r_5 number of tape units in use
- r_6 number of rollouts experienced so far

The value of QP is then a linear combination

$$QP = \sum_{i=1}^{5} R_i(r_i)$$

where

$$R_1(r_1) = 2^6 * r_1$$

$$R_2(r_2) = |\, r_2 - 150{,}100 \,|/128$$

$$R_5(r_5) = \begin{cases} 0 & \text{if } r_5 = 0 \\ 300 + 128\,|\, r_5 - 1 \,| & \text{if } r_5 \geq 1 \end{cases}$$

$$R_6(r_6) = r_6$$

and R_3 and R_4 are shown in Fig. 5. This function QP involves about 22 coefficients.

4.3 An Approach to the Selection of the "Best" Scheduler

We now consider algorithms that involve three features of the configuration of computer runs:

- f_1 number of short jobs (with 30 seconds or less CPU time estimate)
- f_2 remaining number of jobs
- f_3 number of active terminals (which may be used in a variety of modes)

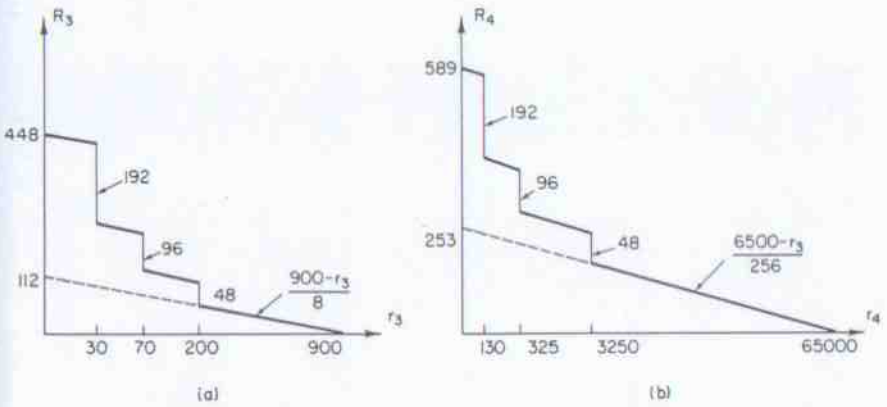


FIG. 5. Graphs of the function $R_3(r_3)$ and $R_4(r_4)$. The horizontal axes are not drawn to scale. Each function is linear plus three step functions. (a) Priority contribution for CPU time; (b) priority contribution for I/O units.

In addition, we use the six-job parameters $\mathbf{r}$ given above and compute queue priority as

$$QP = \sum_{i=1}^{6} R_i(r_i)$$

where

$$R_1(r_1) = a_1 * r_1$$

$$R_2(r_2) = a_2 * (150{,}100 - r_2)$$

$$R_3(r_3) = a_3 * \max(a_4 - r_3, 0) + a_5 * \mid a_6 - r_3 \mid_+^0 + a_7 * \mid a_8 - r_3 \mid_+^0$$

$$R_4(r_4) = a_9 * \max(a_{10} - r_4, 0) + a_{11} * \mid a_{12} - r_4 \mid_+^0 + a_{13} * \mid a_{14} - r_4 \mid_+^0$$

$$R_5(r_5) = a_{15} * \mid r_5 - a_{16} \mid_+^0 + a_{17} * \mid a_{18} - r_5 \mid_+^0$$

$$R_6(r_6) = a_{19} * r_6$$

Recall the notation

$$\mid x - c \mid_+^n = \begin{cases} (x-c)^n & \text{if } x \geq c \\ 0 & \text{if } x \leq c \end{cases}$$

This queue priority function is a slightly modified and simplified version of the actual one above.

We choose a three-dimensional performance measure space with $\mathbf{p} = (p_1, p_2, p_3)$, where

p_1 (mean internal processing time for short batch jobs)/1000
p_2 (mean internal processing time for other batch jobs)/4000
p_3 (mean response time for standard short on-line tasks)/10

The scaling implies that $\mathbf{p} = (1,1,1)$ corresponds to approximately a 15-minute average processing time for short batch jobs, a 1-hour average processing time for other jobs, and a 10-second response time on-line. The algorithm performance is then measured by

$$\| \mathbf{p} \| = w_1 p_1 + w_2 p_2 + w_3 p_3$$

where $\mathbf{w}$ is from the three-dimensional criteria space with $w_i \geq 0$ and $w_1 + w_2 + w_3 = 1$.

The situation for determining the coefficients of the scheduling algorithm is as follows:

1. The computer operator selects a criteria vector $\mathbf{w}$.
2. The operating system measures the configuration features f_1, f_2, f_3.
3. The appropriate best coefficients $\mathbf{a}$ are used for these values of w_i

Thus we see that the 19 coefficients are, in fact, functions of six other independent variables. One could, for example, attempt to determine coefficients α_{ij} so that

$$a_i = \alpha_{i0} + \sum_{j=1}^{3} (\alpha_{ij} f_j + \alpha_{i,j+3} w_j)$$

There is no *a priori* reason to assume this linear relationship is appropriate, but it might be and it is simple. It leads then to 133 coefficients α_{ij}, $i = 1$ to 19, $j = 0$ to 6, for the algorithm selection problem.

It is appropriate to question the validity of this form of the scheduling algorithm from the point of view of the intrinsic complexity of the problem. Such a consideration is entirely subjective at this point because no one has made a thorough analysis of this problem. It seems intuitively plausible that the complexity of this scheduling algorithm form is somewhat high. That is, considering the number of variables involved and the desired precision of the scheduling, it is likely that an adequate form exists with perhaps 40–70 independent coefficients. A crucial point is that (at this time) not enough is known about the effect of scheduling algorithms on system performance for one to identify the really concise, yet adequately precise, forms for scheduling algorithms.

We now consider how to find the best scheduler of this form. To set the context, let us outline how the computation might go in an ideal world. The basic building block would be the computation of best a_i for given w_j and f_j. This block is designated by the function OPT, i.e., OPT$(\mathbf{w},\mathbf{f})$ is the set of 19 best coefficients. Note that this does not involve any assumption about the form of the relationship between the a_i and the variables w_j and f_j, i.e., the α_{ij} are not involved. We would then select an appropriate set of values for the variables w_j and f_j, say w_{jl}, $l = 1$ to m_w, and f_{jk}, $k = 1$ to m_f, and execute the algorithm

$$\text{For } \quad l = 1 \text{ to } m_w, \quad k = 1 \text{ to } m_f \quad \text{do} \qquad a_i(l,k) = \text{OPT}(\mathbf{w}_l, \mathbf{f}_k)$$

At this point we now have a tabulation of the coefficients a_i as a function of the w_j and f_j. The final step is to do a linear least squares fit to obtain the final coefficients α_{ij}.

Let us consider ways that this simple-minded computational approach may go wrong. We list some obvious ways (no doubt there are others waiting if one actually tries the approach):

(1) The function OPT is too difficult to compute. We would say that 50–200 evaluations of functions (that is, $\mathbf{p}$ as a function of $\mathbf{a}$) should be considered reasonable. More than 500 or 1000 indicates real difficulties and less than 50 real luck

(2) The form chosen for QP as a function of **a** is inadequate. This is not likely since the form is the one in current use.

(3) The linear form for the **a** as a function of the w_j and f_j is inadequate.

(4) One is unable to vary f_1, f_2, and f_3 over the range of values as indicated in the system, and thus they are dynamically varying and uncontrollable. To create configurations with known features is probably a very substantial task.

(5) The measurement of $\| \mathbf{p} \|$ is uncertain due to the dynamic nature of the process. That is, in the 15 minutes that it takes for a batch job to go through the system there may have been wide variations in the values of **f** (due to the changing job configuration) and the values of **a** (due to changes made by OPT).

We note that difficulties 2 and 3 are from the problem formulation and not the computation, so we ignore them here. The difficulty with OPT might be very real, but one can be optimistic that a good minimization polyalgorithm will handle this part of the computation—especially after some experience is obtained so that good initial guesses are available. This leaves difficulties 4 and 5, which are very interesting and somewhat unusual in standard optimization problems.

It seems plausible that one can obtain values of $\| \mathbf{p} \|$ that are fairly tightly associated with values of **w**, **f**, and **a**. This means that it is, in principle, feasible to carry out the optimization problem. A simplified example of the situation is shown in Fig. 6, where we assume there is one variable for **a**, and one variable for **w** and **f**.

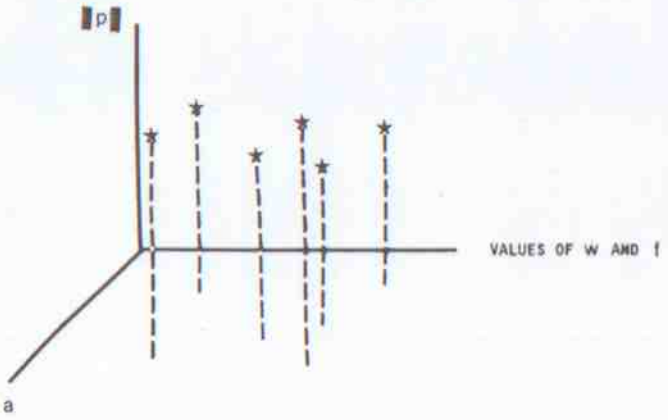


FIG. 6. Function values of $\|\mathbf{p}\|$ obtained when there is no direct control over some of the arguments (**f** in this case).

In order to compensate for the irregular nature of the values obtained, one should use an integral form of the minimization problem and then introduce quadrature rules to accommodate the irregularity. Standard quadrature rules for this situation are not available. Reasonable accuracy can be achieved by using ordinary Reimann sums with areas determined from a space-filling curve map. That is, one maps the high-dimensional domain onto [0,1], then one assigns weights to the points according to the length their images span in [0,1]. Note that certain values of **f** might be very uncommon and hence the optimization obtained there might be unreliable. Fortunately, the rarity of these values of **f** means that the reliability of the scheduling algorithm in that domain is not so crucial. In summary, it appears that adequate methods probably can be found to carry out the computational approach outlined earlier.

As a final note, we consider the time that might be required for a complete determination of the "best" scheduling algorithm. Given a fairly constant job configuration, we assume that we can obtain values for $\| \mathbf{p} \|$ and all other quantities within a 10-minute time interval. This corresponds to one function evaluation. Thus we are led to assume that one evaluation of OPT takes from $\frac{1}{2}$ to 1 day of system time. The inefficiency due to the lack of control over setting parameters will probably double this time, say to $1\frac{1}{2}$ days. The number of evaluations of OPT needed to obtain semi-reasonable reliability in the α_{ij} computations is probably of the order of 50 or 100. This implies about 3 to 6 months to select the best scheduling algorithm.

Note how this approach differs from the common theoretical approach. There one assumes some model for the computer operation and then analytically obtains a good (or optimum) scheduling algorithm for that model. Here there is no explicit model of the computer operation; one tries to obtain a good scheduling algorithm by observing the systems behavior directly rather than through the intermediary of a mathematical model. It is, or course, yet to be seen just how feasible or effective this direct approach will be.

It is obvious that the determination of the best scheduler by this means involves a substantial investment of effort. One has very little feel for the possible payoff from obtaining the best scheduler. It might be nil if the system efficiency is determined by bottlenecks located elsewhere. It might be very large if the scheduler is one of the bottlenecks. One way to estimate the possible gain would be to make a system simulation and perform the optimization there. This would still be a substantial project and would only give an estimate of the possible gain in optimizing the scheduler. Nevertheless, it appears that it might be wise to do this simulation before

attempting to involve a running computer system. Finally, we note that the algorithm selection approach described here can be applied to the other resource scheduling tasks (disk access, communications controllers, etc.) in the same way.

5. Discussion of the Two Concrete Applications

The purpose of the preceding two sections was to examine concrete problems within the framework of the abstract models developed. Our main objective is to examine these two problems and not to solve them. We do, however, given an outline of how a realistic selection might proceed. These concrete examples show the diversity of real problems that fit into the abstract models. We might have also included an examination of a function evaluation problem [e.g., SQRT(X) or SIN(X)], but that seems rather dull since such selection problems have been analyzed in great detail by others.

The two problems considered here have some characteristics in common:

(1) They are real problems subject to active research.

(2) The problem space for the algorithms is of high dimension and the overall nature of the problem is not too well understood. One concludes that the selection problem is essentially complicated by this high dimensionality.

(3) Performance criteria are somewhat subjective and vary considerably from context to context.

(4) The algorithms involve familiar mathematical functions and the algorithm selection problem can be formulated as a more or less standard (though complicated) mathematical approximation problem.

There are also some large differences in the characteristics of these two problems:

(5) There is a substantial body of relevant data available for the quadrature problem, but nothing for the scheduling problem. The data for the quadrature problem have not been collected systematically and are thus less useful than one might hope.

(6) The scheduling algorithm involves a complex dynamic process in such a way that:

- (a) some independent parameters cannot be varied at will;
- (b) reproducibility of results is unlikely since one rarely has the same element of the problem space twice;
- (c) large amounts of calendar time are required for the selection of "best" algorithms.

6. Approximation Theory Machinery

6.1 Formulation and Structure of the Approximation Problem

The purpose of this section is to analyze the algorithm selection problem within the framework of approximation theory. We will see that the principal questions of this problem can be formulated within the traditional framework of approximation theory. Even so, the answers to many of the questions require the development of very novel techniques and theories of approximation. More specifically then, our purpose is systematically to examine these questions, to indicate what light can be shed on them from the existing theory of approximation, and to point out the new problems in approximation theory that are raised by the algorithm selection problem. Needless to say, we do not propose to solve these new problems here. The principle questions are divided into four groups:

(1) norms and approximation forms,
(2) degree of convergence, complexity, and robustness,
(3) existence, uniqueness, and characterization,
(4) computation.

The question of computation is not considered in this chapter since it would seem to involve half the known methods of computation.

6.2 Norms and Approximation Forms

The question of norms enters in the final step from the algorithm performance space $\mathfrak{R}^n$ to the single number that represents the algorithm performance. Since we have a norm on a standard n-dimensional vector space, the possibilities are well known. The most common are of the form

$$\| p \| = \left[\sum_{i=1}^{N} w_i p_i^r \right]^{1/r}$$

with typical values of r being 1, 2, or ∞ (for the Tchebycheff or minimax norm). However, the nature of the selection problem is such that we can anticipate using nonstandard norms. The reason is that the performance measures tend to include essentially incomparable variables, e.g.,

p_1 computer time used (measured in seconds)
p_2 computer memory used (measured in words)
p_3 complexity of setting up the computer run (measured in hours required by the programmer)

A plausible norm to use in such a context might be

$$\| p \| = p_1 + \alpha(p_2) p_2 + \beta(p_3) p_3$$

where

$$\alpha(p_2) = \begin{cases} 0 & \text{for} \quad p_2 \le 10{,}000 \\ 10^{-5} & \text{for} \quad 10{,}000 \le p_2 \le 20{,}000 \\ 2*10^{-5} & \text{for} \quad 20{,}000 \le p_2 \le 30{,}000 \\ 7p_2*10^{-9} & \text{for} \quad p_2 \ge 30{,}000 \end{cases}$$

$$\beta(p_3) = \begin{cases} 0 & \text{for} \quad p_3 \le 0.5 \\ 2 & \text{for} \quad 0.5 \le p_3 \le 2 \\ p_3 & \text{for} \quad p_3 \ge 2 \end{cases}$$

There are two observations, one positive and one negative, about such complicated norms that can be made based on current experience in approximation. The negative one is that they do complicate the theory sometimes and more often make the computations substantially more difficult. The positive one is that the choice of norm is normally a secondary effect compared to the choice of approximation form. That is, if one has a good choice of approximation form, one obtains a good approximation for any reasonable norm. This implies that one can, within reason, modify the norm used so as to simplify the analysis or computations. A significant corollary to this last observation is that one cannot compensate for a poor choice of approximation form by computing power or technical skill in analysis.

We now turn to the crucial question of approximation forms, which we group into five classes: (a) discrete, (b) linear, (c) piecewise, (d) general nonlinear: standard mathematical, separable, abstract, and (e) tree and algorithm forms.

In order to discuss these choices, we need to formulate more precisely the standard idea of approximation form as it currently exists in approximation theory. The form is to be used for the selection mapping $S(f(x))$: $\mathfrak{F} \to \mathfrak{A}$ and we visualize a parameter (or coefficient) space $\mathfrak{C}$ plus a particular form of the mapping. To show explicitly the dependence of S on the coefficients, we may write $S(f(x),c)$ at times. Specific examples of the five classes of approximation forms are given below:

(a) *Discrete*

$$S(f(x),1) = \text{computer program 1}$$
$$S(f(x),2) = \text{computer program 2}$$
$$S(f(x),3) = \text{computer program 3}$$

(b) *Linear*

$$S(f(x),c) = c_1+c_2f_1+c_3f_1^2+c_4(f_1f_2)^2+c_5(f_2-f_3)^3+c_6/f_3$$

Note that linear refers to the dependence on the coefficients c_i and not the features f_j.

(c) *Piecewise linear*

$$S(f(x),c) = \begin{cases} c_1+c_2f_1+c_3f_2+c_4f_1f_2+c_5/f_2 & \text{for} \quad |f_1+f_2| \ge 2 \\ c_6+c_7f_1+c_8f_2+c_9f_1f_2+c_{10}f_1^2 & \text{for} \quad |f_1+f_2| \le 2 \quad \text{and} \quad f_1 \le f_2 \\ c_{11}+c_{12}f_1+c_{13}f_2+c_{14}\dfrac{f_1-f_2}{1+f_1+f_2} & \text{for} \quad |f_1+f_2| \le 2 \quad \text{and} \quad f_1 \ge f_2 \end{cases}$$

We see that the feature space is subdivided into pieces and $S(f(x),c)$ is defined linearly on each of the pieces.

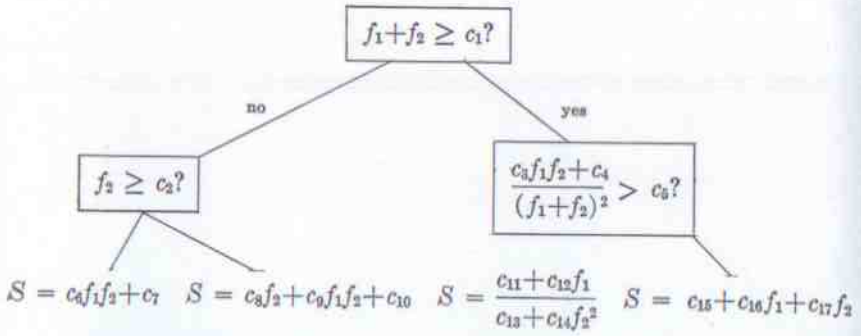
(d) *Nonlinear, standard forms*

$$S(f(x),c) = \begin{cases} \dfrac{c_1+c_2f_1+c_3f_2}{c_4+c_5(f_1+f_2)^2} & \text{(rational)} \\ c_1e^{-c_2f_1}+c_3e^{-c_4f_2}+c_6e^{-c_8(f_1+f_2)^2} & \text{(exponential)} \\ c_1+c_2f_1+c_3f_2+c_5(f_1-c_4)_+ +c_7(f_2-c_6)_+ & \text{(spline)} \\ \qquad \text{where} \quad (f-c)_+ = \begin{cases} 0 & \text{for} \quad f \le c \\ f-c & \text{for} \quad f \ge c \end{cases} \\ g_1(f_1,c_1,c_2)+g_2(f_2,c_3,c_4)+g_3(f_3,c_5,c_6) & \text{(nonlinear, separable)} \end{cases}$$

The spline example has variable pieces. If c_4 and c_6 were constants, then this would be piecewise linear. For the nonlinear separable case the effects of the different features (and their associated coefficients) are completely independent of one another. The exponential example given just above is also of this form.

The abstract nonlinear form is an arbitrary function of the features $f(x)$ and the coefficients c.

(e) *Tree and algorithm forms:*




```
      FUNCTION  S(F,C)
      SUM=0
      DO 20  K=1,  C(1)
20        SUM=SUM+C(K+1)*F(K)
      IF( F(1) > C(1) ) THEN SUM = SUM/( C(C(1)+1))+1 )
      PROD=1.
      IF( F(C(1)+2) < (C(C(1)+1)+F(2))/F(3) )
          THEN PROD=F(1)*F(2)
      DO 40 K=1, C(C(1)+3)
40        PROD = ( F(K)+C(K))*PROD+C(C(1)+K+3 )
      S = C(1)*SUM+C(2)*PROD+C( C(1)+C( C(1)+3)+1)*F(1)
```

The main thrust of approximation theory is for the case where the coefficients c are used to parameterize a relatively simple form (i.e., such as the linear, piecewise linear, and nonlinear forms). The distinguishing characteristic of these cases is that the set of approximation forms can (at least locally) be identified with a manifold in some ordinary finite-dimensional space. The approximation theory machinery is then used to obtain the best coefficients or parameters (again, at least locally) from this manifold.

One thus may conclude that there are three distinct situations as far as the applicability of approximation theory machinery. The first and most favorable situation is for the linear, piecewise linear, and nonlinear approximation forms. Here the machinery may be applied essentially as it currently exists. This does not mean that all of these cases are already solved and all one has to do is to "copy" the solutions from somewhere. Rather, it means that these are the kinds of problems the machinery is supposed to handle and, if it is currently inadequate in some specific instance, it needs to be extended in the direction it is already headed.

The second situation is for the tree and algorithm forms. Here it seems

that a major change in emphasis is required. The exact nature of the new machinery is certainly unclear and no doubt there are hidden difficulties not apparent from a casual inspection. However, it seems plausible that the general spirit of the approach and techniques may well be similar to that already existing. For example, the piecewise linear forms may be visualized as one of the simplest of the tree forms. The development and analysis for the piecewise forms (even for variable pieces) has progressed fairly smoothly over the past 10 years and the resulting body of results has very much of the flavor of the previously established linear and specialized nonlinear theories. There were (and still are), of course, some difficult questions for the piecewise linear, but the prospects do not appear to be too bad for developing a useful body of approximation theory machinery for the tree and algorithm forms.

The third and least favorable situation is for the discrete forms. The standard mathematical approach results in stating that the problem is trivial in this case. One ascertains the best selection mapping by a finite enumeration. Unfortunately, the enumeration may well be over very large sets. Even 1000 elements (algorithms) are completely unmanageable in most instances and it is easy to find problems where there are millions of algorithms to be considered (at least in some abstract sense). It is not at all clear how algorithm selection procedures are to evolve in this situation and the development of such procedures is one of the foremost open questions in this entire area of study.

We close this section by repeating a fundamental observation: *The most important single part of the successful solution of an approximation problem is the appropriate choice of the approximation form.* Approximation theory machinery comes into play *after* this choice is made. Thus it is essential to have insight into both the problem and algorithm spaces and into the possible forms one might choose for the selection mappings.

6.3 Classification of Problems, Degree of Convergence, Complexity, and Robustness

This section has two distinct parts. First, we introduce the concept of classifying problems, and second, we introduce three other concepts that are intimately related to ways of classifying problems. These three concepts—degree of convergence, complexity, and robustness—are important for evaluating the overall value of various approximation forms for the algorithm selection problem.

6.3.1 *Classification of Problems*

An important approach to obtaining insight into the nature of the problem space is to partition it into particular classes of problems. Ideally there

is a representative member or property of each class that is especially relevant to the selection of algorithms. The exact nature of the classification depends, of course, essentially on the specific problem space. Some typical examples include the following.

(a) Numerical quadrature: Compute $If = \int_a^b f(x)\,dx$.
Class 1: Those $f(x)$ which have continuous curvature.
Class 2: Those $f(x)$ which have five or fewer oscillations in $[a,b]$.
Class 3: Those $f(x)$ which are analytic.

Mathematics has a highly developed classification system for functions [integrands $f(x)$], which provides literally dozens of classes relevant to numerical integration algorithms.

(b) Scheduling a CPU in an operating system.
Class 1: Batch processing multiprogramming, one CPU, two I/O channels, and one disk.
Class 2: Time sharing, two CPU's, 50 terminals.
Class 3: Time sharing with a batch processing background, two CPU's, 50 terminals, saturation loading.

We see that the problem classification has many independent variables giving a high-dimensional problem space.

(c) Scene analysis.
Class 1: One connected object, a line drawing with 50 or fewer lines.
Class 2: Up to 10 objects, each composed of from 1 to 10 rectangles, triangles, or circular arcs.
Class 3: Unknown number of separated objects of one of four types; distinguishing properties are color, texture, size, position, and orientation.

It is easy to visualize thousands of particular types of scenes to analyze.

The idea of problem classification is simple, but important. Most algorithms are developed for a particular class of problems even though the class is never explicitly defined. Thus the performance of algorithms is unlikely to be understood without some idea of the problem class associated with their development.

It is particularly common to attempt a classification system that goes from easy to hard. Thus one visualizes a nested set of problems where the innermost set consists of very easy problems and the largest set consists of very hard ones. Unfortunately, it is not always easy to make such a classification (at least in a reasonable way) for complex problem spaces. One is lacking the insight to know in all circumstances just what makes a problem hard or easy.

6.3.2 Degree of Convergence

The idea of degree of convergence comes from considering a sequence of approximation forms and asking: How much better do these forms do as one goes further out in the sequence? A standard example would be for computing $\log x$ by polynomials of degree $0,1,2,3,\ldots,N,\ldots$. We assume that for each approximation from the sequence we have the best coefficients possible.

In the present context, our ultimate objective is to choose the best algorithm for every problem. If we let $A^*(x)$ be the best algorithm for problem x and $A_N(x)$ the algorithm chosen by the best coefficients for the Nth approximation form, then the question is: *How does*

$$\epsilon_N(x) = \| p(A^*(x)) \| - \| p(A_N(x)) \|$$

behave as N gets big? Does it go to zero for every x? Suppose we set

$$\epsilon_N = \max_{x \in \mathcal{P}} \epsilon_N(x)$$

Does ϵ_N go to zero fast, slowly, or at all? The answer to these questions is called the *degree of convergence* for the problem space $\mathcal{P}$ and the sequence of approximation forms.

In standard mathematical situations this idea is well developed and the degree of convergence is known for many cases. In the standard case the problem is to evaluate a function $f(x)$ and the best algorithm $A^*(x)$ is taken to be the exact value of $f(x)$. The measure of performance of an algorithm A that produces an approximation $a(x)$ is taken to be $|f(x) - a(x)|$. Thus, for computing $\sin(x)$ for $x \in \mathcal{P} = [0,\pi/2]$ we know that polynomial forms give $\epsilon_N \sim KN^{-N}$ for some constant K. In this case, ϵ_N goes to zero extremely fast. If one replaces $\sin(x)$ by $\mathrm{ABS}(X-1)$, then $\epsilon_N \sim KN^{-1}$, which is not very fast at all.

The analogy with approximately evaluating a function can be carried further, but theoretical information about the degree of convergence is limited to "mathematical" functions, that is, functions defined in a mathematical context where one knows a variety of properties. We can say, however, that really fast convergence using simple forms (i.e., polynomials and similar linear forms) *requires* that the function involved be very well behaved. By well behaved we mean smooth (no jumps or discontinuities any kind, including in derivatives) and of a consistent global nature (i.e., if it oscillates one place, it oscillates everywhere, if it is flat one place, it is flat everywhere). A large proportion (at least 50%) of the "functions" that arise naturally in the real world are *not* well behaved in this sense.

6.3.3 *Complexity*

A fashionable idea related to degree of convergence is complexity. Thus the complexity of a function is some intrinsic measure of how hard it is to compute the function. The idea extends directly to solving problems by noting that solving a problem is equivalent to computing the value of the function that gives the solution of the problem.

In actually measuring complexity, one does several things:

A. Introduce some measure of the work involved in a computation. Typical examples are number of arithmetic operations, number of multiplies, execution time of a real program on a particular real computer, length of FORTRAN program needed, number of steps in a Turing machine computation.

B. Assume that one considers the most efficient scheme. There is no limit on how badly one can evaluate a function; complexity is measured with methods of optimal efficiency.

C. Restrict the kinds of steps in the algorithms used for the computation. For example, polynomial approximation excludes division, so $1/x$ may be difficult to compute, but if division were allowed then this would be a very easy function. Similarly $|x-0.5|$ is very easy if ABS is allowed or if a test and branch operation is allowed.

A uniform way to impose the above conditions on the complexity question is to say that the function is to be evaluated by a particular machine or, essentially equivalent, by one of a particular class of programs for a general purpose machine. We illustrate this approach for polynomials in Fig. 7.

The advantage of the idea of complexity over that of the degree of convergence is that much greater generality is achieved. Degree of convergence can be normally interpreted as complexity using a very specialized machine. For example, a machine that can only add and multiply but that can be programmed to do this in more or less arbitrary sequence and with arbitrary operands is considerably more versatile than the polynomial evaluation machine shown in Fig. 7. It could, for example, evaluate the function X^{1024} in 10 operations rather than the 1024 required for the strictly limited polynomial evaluation machine. This added generality also makes it possible to place the standard mathematical approximation forms into the same framework as the piecewise forms and the tree or algorithm forms. One merely adds or changes a piece of "hardware" on the machine.

The disadvantage of the idea of complexity is that its generality makes it very difficult to obtain specific results. Current research is very intensive and yet concentrated on rather simple problems as seen in Table I. These

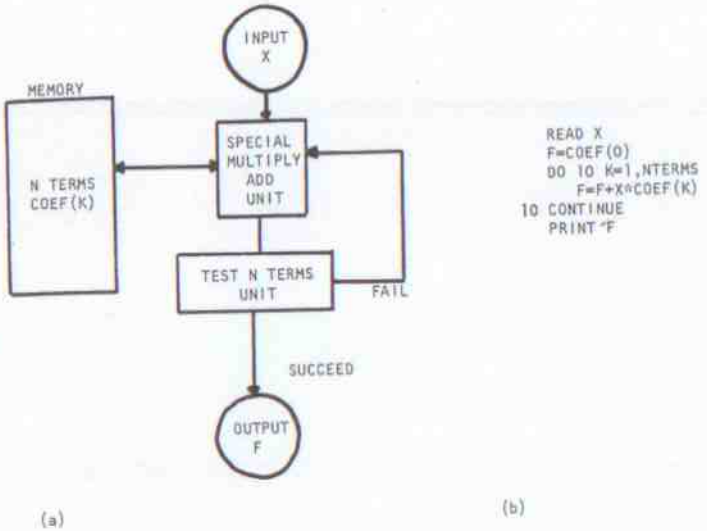


FIG. 7. Polynomial evaluation via (a) machine or (b) program. The special MULTIPLY/ADD unit and TEST unit of the machine are such that they can only (and automatically do) execute the program on the right.

problems are orders of magnitude simpler than the typical situation that arises in the algorithm selection problem. Thus there is little hope for the near future that we will obtain optimal algorithms for most of these problems (except possibly from *very* limited subclasses of algorithms).

In spite of the low probability of obtaining precise results about complexity in the algorithm selection problem, there are three good reasons to consider the idea. First, it provides the proper framework within which to

TABLE I

SUMMARY OF COMPLEXITY RESULTS FOR SOME COMMON COMPUTATIONS

Computation	Work or complexity of		
	Standard method	Optimal	Best known
Add two N-digit integers	N	N	N
Multiply two N-digit integers	N^2	?	$N \log^2 N$
Evaluate polynomial of degree N	N multiplies	?	$[N/2]$ + 2 multiplies
Median of list of length N	$N \log N$	N	N
Multiply two $N \times N$ matrices	N^3	?	$N^{2.7}$ ...

contemplate the problem. Second, the results for simple problems show that the standard ways of doing things are often not optimal or even anywhere close to best. Third, the high degree of complexity in "real" problems indicates that simple-minded approaches are unlikely to do well and even sophisticated approaches will often fall very short of optimal. Indeed, it is likely that further theoretical developments in the area will indicate that it is essentially impossible to obtain the optimal algorithms for many real problems.

6.3.4 *Robustness*

Robustness is a technically precise term in statistics that relates the quality of statistical estimates in extreme situations. Thus an estimation procedure is robust if its quality degrades gracefully as the situation becomes more and more extreme. We do not attempt to define this concept precisely here but it is quite useful in considering the selection of algorithms. It is a common phenomena for algorithms to do very well on a certain class of "easy" problems and to do increasingly worse as one moves away from these easy problems. A robust algorithm, then, is one whose performance degrades slowly as one moves away from the problems for which it was designed. Since the problem space is so large and so poorly understood in many real situations, this quality can be extremely important. There is a reasonable probability that one will face a problem with a completely unforeseen combination of attributes, which invalidate some of the "working assumptions" used in the development of the algorithm. The worst situation is, of course, an algorithm that fails completely and quietly as soon as one moves away from the ideal problems.

Consider the simple example of estimating the wealth of the "typical" student in a classroom. One has three candidate algorithms for the estimate: the average wealth, the medium wealth, and the midrange wealth. In a "normal" situation, these algorithms produce similar estimates, any one of which is satisfactory. A difficulty occurs with the presence of Hughes Hunt III (wealth of $625 million) or John D. Mellon V (wealth of $398 million) in the class. The midrange now produces ridiculous estimates like $200 or $300 million and the average is not much better with estimates like $20 or $30 million. The median estimate is, however, essentially unaffected by the presence of such a wealthy person and thus is a very robust algorithm for this problem. While the average is more robust than the midrange, it is not very satisfactory in extreme situations.

Finally, we note that robustness is frequently difficult to identify or measure. In some situations one can achieve robustness with very simple algorithms. In others it seems that robustness requires a complex algorithm that has numerous tests for special situations and cases.

6.4 Brief Survey of Approximation Form Attributes

This section presents a survey of the general attributes of five important types of approximation forms. Of necessity we speak in generalities, and thus there is a real danger that a casual reader will be misled. The statements we make about attributes apply "usually" or "commonly." Realistic specific situations exist that exhibit behaviors exactly opposite to the usual one. We have already noted that the most crucial decision in the algorithm selection problem is that of the approximation form. Ideally, this process goes as follows: One is intimately familiar with the problem space and with a large variety of approximation forms. One weighs the various advantages and disadvantages of the forms as they interact with the special features of the problem space. Perhaps some simple experimentation is made. Finally, a choice of form for the algorithm selection mapping is made that achieves a good balance with the overall objectives.

Thus one can visualize this section as a primer on the choice of approximation forms. Unfortunately, it is only an elementary primer and there is no substitute for detailed experience with a variety of real situations.

6.4.1 *Discrete Forms*

One might tend to dismiss this case as "degenerate." After all, if one is merely to select the best one of three or 11 algorithms, there seems to be little need for any elaborate machinery about approximation forms. We do not imply that identification of the best will be easy; rather we say that concepts like complexity, degree of convergence, etc., do not play a role. This reaction is appropriate in many cases. However, sometimes there are some very interesting and challenging features of these forms.

The principal feature is that the finite number of algorithm is either in fact or in concept a *very large* set. Even though we may have selected just three algorithms, we often visualize that these are representative samples from a very much larger set. Recall from the discussion of the numerical quadrature problem that there may well be tens of millions of algorithms of even a rather restricted nature. Thus in the mind's eye there is almost a continuum of algorithms, even though we may in fact be examining only three of them. One of the major weaknesses of modern mathematical machinery is in its ability to handle problems involving very large finite sets. The emphasis has been on developing tools to handle problems with

infinite sets (e.g., the continuum) and one frequently draws a complete blank when faced with a finite set of, say, 10^{123} elements.

We are really saying that the proper way to consider discrete forms is as a discretization of a continuum. One then applies some intuitive ideas about continuous forms (such as presented later in this section) and hopefully obtains satisfactory results.

Unfortunately, we cannot continue a meaningful discussion here along these lines because we have no knowledge of the possible continuum behind the discrete set.

We conclude by recalling that robustness is a property of individual algorithms and thus immediately relevant to discrete forms. It could be evaluated for each algorithm in the discrete set. However, if the set is large, then this is impractical. In this latter case, one probably must attempt to transfer information about robustness from some underlying continuum.

6.4.2 *Linear Forms*

There are so many obviously nice things about linear forms that we might tend to concentrate too much on what is bad about them, or we might tend to ignore anything bad about them. Some of these nice things are

They are simple and efficient to use.
They are the easiest to analyze (by far).
They are easy to understand and visualize intuitively.
They are often extremely successful in achieving good approximation.

These observations imply that we should give these forms first consideration and that we should try other things only after we are fairly sure that some linear form does not suffice.

The bad thing about these forms comes from the following experimentally observed fact: *Many real world processes are not linear or anywhere close to being linear.* In particular, we would like to emphasize: *Most of the world processes are not a linear combination of simple, standard mathematical entities.* Since these facts are experimental rather than theoretical, we cannot prove them here. Indeed, certain theoretical results (e.g., the Weierstrass theorem) are frequently used to support just the opposite conclusion (e.g., one can use polynomials for everything).

Let us illustrate the situation by a trivial example: Our problem space $\mathcal{P}$ has just one attribute of consequence and we call it x (which identifies the problem with a real number that measure this attribute). Our algorithm space $\mathcal{A}$ is likewise simple with one attribute, which we call A. Suppose that x and A range between 0 and 1 and suppose the best algorithm

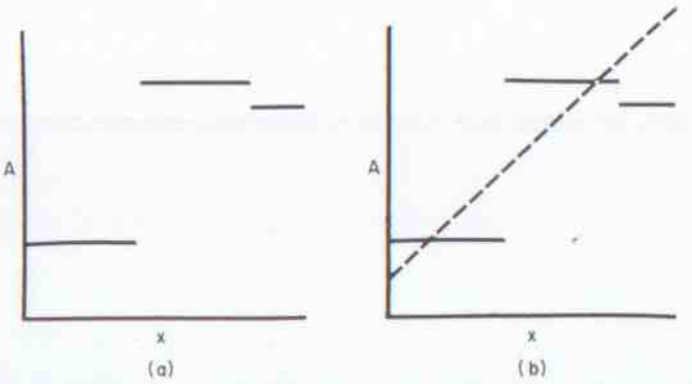


FIG. 8. (a) Graphical representation of the optimal algorithm selection mapping for a simplified example. (b) The optimal plus the best linear algorithm selection mapping.

is for $A = 0.27$ if $x < 0.41$, $A = 0.82$ if $0.41 \leq x \leq 0.8$, and $A = 0.73$ if $x > 0.8$. The best or optimal algorithm selection mapping is then as shown in Fig. 8a. If we attempt a linear form then we would have $A = \alpha + \beta x$, where α and β are coefficients to be determined. The optimal values α^* and β^* for these coefficients give a mapping shown as the dashed line in Fig. 8b. This mapping is clearly not very close to being optimal.

Once this completely linear form is recognized as inadequate, one then tends to proceed on to something more flexible. A natural idea is to use polynomials, e.g.,

$$A = \alpha_1 + \alpha_2 x + \alpha_3 x^2 + \alpha_4 x^3 + \cdots + \alpha_N x^{N-1}$$

If one carries this out for $N = 4$ (cubic polynomials) and $N = 20$, one can expect results such as shown in Fig. 9 (provided one has been careful in the computations). It is hard to argue that either one of these selection mappings is a good approximation to the optimal one. Note that in both cases that the polynomials are truncated at either $A = 0$ or at $A = 1$ in order to avoid obtaining nonexistent algorithms for some values of x.

Can one hope to do much better by choosing something besides polynomials? One frequently sees Fourier series (sines and cosines), exponentials, Bessel functions, etc. None of these give noticeably better approximations. There is, of course, a way to obtain excellent results by a linear form, $A = \alpha + \beta\omega(x)$. By merely choosing $\omega(x)$ to be the optimal selection mapping, then we find $\alpha^* = 0$ and $\beta^* = 1$ gives a perfect approximation.

This last observation shows the impossibility of making universal judgments about linear forms. If one chooses linear combinations of the right things, then the linear forms can do very well indeed. In practice though, one is usually limited to just a few possibilities and one has very little in-

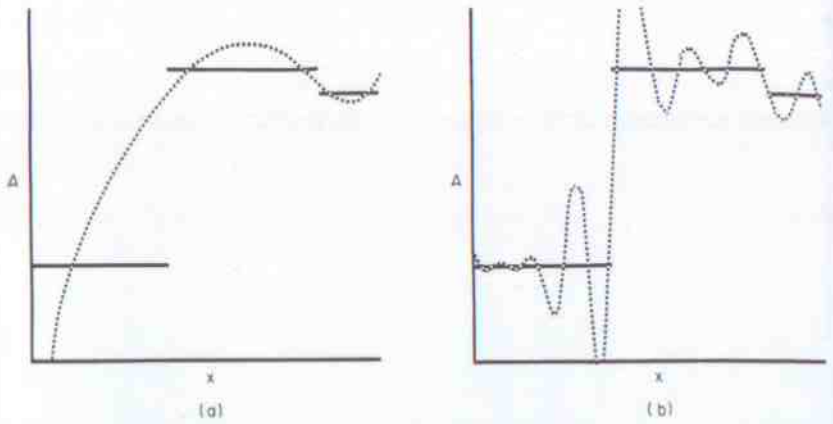


FIG. 9. Graphical representation of the optimal plus the best cubic (a) and best 20th degree (b) polynomial selection mappings.

formation about the optimal mapping. Note that a typical real problem has 5 to 15 dimensions in each of x and A variables. One is not likely to hit upon the optimal mapping as one of the things to include in the linear mapping.

We now attempt to motivate the above conclusions from the point of view of degree of convergence and complexity. For standard mathematical situations there are numerous results about how the errors of polynomial and similar functions behave as the number of terms increases. The phenomenon of Fig. 9 shows very slow convergence, or poor degree of convergence. Of course, if the optimal selection mapping has a jump as seen Fig. 8, there will always be a large error at that jump. We also see that the large error at the jump induces large errors everywhere.

If the optimal mapping is continuous but has breaks in the slope, then it is known that the degree of convergence for N-terms is like $1/N$. That means that if one term gives a unit error, then ten terms give a 0.1 error, 100 terms give 0.01 error, etc. This is a very bad situation even for the simplest case of a one-dimensional problem space. Higher dimensions compound this difficulty enormously. Thus if several of these breaks occur in a K-dimensional problem space, then the error behaves like $1/N^{1/k}$, where N is again the number of terms. For $K = 5$, if 1 term gives a unit error, then we would expect to need about 32 terms for $\frac{1}{2}$ unit error, 1000 terms for $\frac{1}{4}$ unit error, and 100,000 for 0.1 error. For $K = 10$, the corresponding numbers are 1000, 1,000,000, and 10^{10}, respectively, for errors of $\frac{1}{2}$, $\frac{1}{4}$, and 0.1. Clearly polynomials and related functions are hopeless in such situations except for the crudest of approximations to the optimal selection mapping.

How often can one expect the problem space to produce selection mappings with these troublesome properties? Experimental evidence with phenomena from physics and engineering problems indicates that more than 50% of these functions are unsuitable for polynomials and other standard linear mathematical forms. This includes Fourier series, which are currently widely used in engineering situations where they cannot possibly give accurate results. There is an intuitive reason why one should expect this. Many physical phenomena have several domains where different factors completely dominate the behavior. As one goes from one domain to another there is a kind of discontinuity in behavior even if there is no sharp break in the slope. These discontinuities affect the degree of convergence directly and, expecially for low accuracies, lead to a very excessive number of turns being required. Recall that polynomials, Fourier series, etc., have the property that their global behavior is completely determined by their behavior on an arbitrarily small domain. This property is not present in many real world situations and is another intuitive reason for doubting the general applicability of the standard mathematical forms.

One must admit that the above arguments are taken from simplified and specialized situations. The extrapolation to all kinds of algorithm selection problems is very tenuous indeed. Yet, we conjecture that things get worse rather than better as one gets away from these situations into a broad range of real world problems.

6.4.3 *Piecewise Linear Forms*

In simple terms, we break up the problem domain into pieces and use separate linear forms on each piece. The motivation is to circumvent the difficulties described in the preceding discussion. In many cases the most crucial step is to determine the appropriate pieces, and yet these forms assume that they are fixed and given by some *a priori* process. In these cases we have, in fact, a two-stage process: first, an intuitive, hopefully realistic, partition of the problem domain into separate pieces, and second, the application of mathematical techniques to obtain the best coefficients for each of the linear pieces. Note that there are often some interconnections between the pieces (for example, broken lines are piecewise linear functions of one variable that join up continuously) that give rise to mathematical problems that are nonstandard but still linear (and hence usually tractable.

It is difficult to draw general conclusions about this approach because of the vagueness of the process for determining the pieces. Indeed if the pieces are poorly chosen or too big, then one can have all the difficulties mentioned with the traditional linear forms. On the other hand, there are

the following hopeful facts about this approach:

(1) Sometimes one does have good enough intuition to determine the pieces so that a very significant improvement is made. Sometimes only a very few pieces are required for this improvement to happen.
(2) Sometimes the problem domain is small enough that one can break it up into more or less equal pieces that are small enough to obtain good results and yet still not obtain an intractable number of pieces.
(3) There are theoretical results (admittedly again from the narrow context of approximating functions of one variable) that indicate that if the best selection of pieces is made, then there are fantastic improvements possible, that is, so that the degree of convergence may change from something like $1/N^{1/2}$ to $1/N^3$, where N is the number of coefficients involved. If one piece gives accuracy 1, then these convergence rates indicate that about 10,000 or 5 coefficients, respectively, are needed to give an accuracy of 0.01 in the determination of the best selection mapping. Such an improvement obviously changes the entire nature of the problem.

We conclude that piecewise linear forms merit separate consideration for three reasons:

(1) They are nonstandard in mathematical/scientific analysis and might be overlooked if lumped into a larger class.
(2) Once the difficult determination of pieces is made, then more or less standard machinery can be used in further analysis and computation.
(3) They have been very useful in a variety of difficult situations and, while they are not a panacea, there is reason to believe that they will continue to be very useful.

6.4.4 General Nonlinear Forms

It is not very profitable to discuss such forms in the abstract. These forms include everything, including the best possible selection mapping, and thus one can do perfectly with them. Therefore, we must really be concerned with various specific classes of nonlinear forms. The literature on approximation theory contains a considerable development of a variety of such classes. A partial list of these with simple examples is as follows.

Rational functions:

$$\frac{c_1+c_2x+c_3x^2}{c_4+c_5x}$$

Exponential/trigonometric functions:

$$c_1e^{c_2x} + c_3e^{c_4x} + c_5\cos(c_6x)$$

Piecewise polynomials:

$$\begin{cases} c_1+c_2x+c_3x^2 & \text{for } -\infty < x \le c_4 \\ c_5+c_6x+c_7x^2+c_8x^3 & \text{for } c_4 \le x \le c_9 \\ c_{10}+c_{11}x+c_{12}x^2 & \text{for } c_9 \le x < \infty \end{cases}$$

Unisolvent functions: The set of all conic sections in the place.
Varisolvent functions: A general class of nonlinear forms, which includes the rationals, exponentials, etc.

There are several general statements that one can make about these forms:

(1) A considerable (or even very extensive) amount of analysis has been made of the theory of approximation.
(2) In those cases where degree of convergence results are available (e.g., piecewise polynomials and rationals), they imply that these special forms are much more capable of approximating a wide variety of behaviors. For example, both rationals and piecewise polynomials can do very well at approximating a jump discontinuity or a behavior like $x^{1/2}$ or $1/x^{1/2}$.
(3) The computational effort required to obtain best (or even very good) coefficients of these forms can be substantial. The development of computational methods is more difficult than for linear forms. However, it is practical to carry out these computations in a variety of cases.

Thus one expects (and observes) these forms to be useful in a variety of situations. The key to success is to analyze one's particular situation sufficiently to obtain general knowledge of the required behavior of the selection mapping. One then chooses that nonlinear form which possesses this behavior and for which one can handle the analytical and computational difficulties.

In conclusion, the determination of the proper nonlinear form is still somewhat of an art and there is no algorithm for making the choice. On the other hand, the degree of convergence and complexity results for rational functions and piecewise polynomials show that they have great flexibility and are likely to do well in most situations. Doing well might not be good enough. In real problems the dimensionalities are high, and needing five coefficients per dimension implies that 5^n coefficients are required for an n-dimensional feature (or problem) space. With $n = 2$ this is a modest 25 coefficients, but $n = 10$ would then require almost 10 million coefficients. This 10 million may be considered doing well compared to the 6 decillion coefficients of another approach, but in either case one cannot use the forms.

6.4.5 Tree and Algorithm Forms

These forms are most intriguing because they promise so much and have the mystery of the unknown. Perhaps it is a case of the grass being greener on the other side of the fence. These forms may have difficulties and disadvantages that are not apparent now but that may limit their usefulness much more than one hopes.

The primary basis for their promise is their flexibility and potential for complexity. They certainly should complement the more traditional mathematical forms. Their flexibility and complexity might be the limitation on their application. Computational methods for good coefficients of traditional forms have taken many years to develop and even now can be quite demanding. It may well be that the computation of good coefficients will severely restrict the usefulness of these forms for many years.

The piecewise linear forms are an example of a simple tree form and their success bodes well for other cases. Computational techniques and theoretical analysis for these forms are progressing steadily and we can look for them to enter into the "standard and routine" category before long. This development should serve as a useful guide for other simple tree and algorithmic forms. Still, we are very far removed from the time when we can select as our approximation form a 72-line FORTRAN program and then compute the best "coefficient values" (FORTRAN statements) for a particular application.

In summary, we have very little hard information about these forms, but they appear to hold great promise and to provide a great challenge for theoreticians and practitioners.

6.5 An Error to Avoid

Occasionally one observes the following situation develop:

(1) A real world problem is considered.

(2) A crude model is made of it. This model perhaps has some undetermined coefficients or is to be manipulated to obtain predictions about the real world problem's solution.

(3) A huge effort is spent in obtaining accurate coefficients or predictions based on the model.

In the specific instance at hand, the real world problem is the algorithm selection mapping, the model is the approximation form selected, and the effort is in determining the coefficients of this form. The error that one can make is in believing that finding the best coefficients of the selection mapping will result in good selections. In many cases *there is no reason to believe that the best coefficients will give good selections.* One is particularly

susceptible to making this error when using simple linear forms for the selection mapping. One may refer to Fig. 8 for an illustration of this situation.

6.6 The Mathematical Theory Questions

This section presents an intuitive summary of the three principal topics of approximation theory. The algorithm selection problem presents some new open questions in these topics, some of which are indicated. There is more emphasis on summarizing the theory of approximations than on the implications for the algorithm selection problem.

6.6.1 The Existence Question

In concrete situations one rarely worries about the existence of best selection algorithms (even though one continually worries about the existence of good ones). Yet, from time to time this question sheds important light on practical questions. Parameterization plays an important role here: one is continually identifying algorithms by means of a set of coefficients or parameters. The question of existence of a best algorithm then becomes a question of the existence of a best set of coefficients. In the simplest cases (e.g., linear forms) the coefficients are just sets of real numbers and the question is readily reduced to a problem about sets of real numbers. One then attempts to show that (a) infinite coefficients cannot be best, and (b) the algorithms depend continuously on the coefficients. It then follows from standard mathematical arguments that a best set of coefficients exists.

This line of reasoning may fail at various points for nonlinear approximation forms. The failure is usually because of some weakness in the parameterization. A key point to remember is to *distinguish carefully between an approximation form and the particular set of coefficients used to parameterize it.* Consider the two simple examples

$$S(f,c) = c_1+f/c_2, \qquad S(f,c) = c_1+e^{-c_2 f}$$

In both of these cases $c_2 = +\infty$ corresponds to a constant and hence a perfectly reasonable function. In the first example this is due to a silly parameterization; one should have $c_1+c_2 f$ instead. It is sometimes not so easy to see such silliness in more complex examples. The second example presents a more delicate situation, as there is no familiar mathematical way to rewrite this form so that the difficulty disappears. One can, however, obtain a perfectly satisfactory parameterization by taking c_1 and c_2 to be the values of $S(f,c)$ at $f = 0$ and $f = 1$, respectively. However, there is now no nice way to express $S(f,c)$ explicitly in terms of c_1 and c_2.

True nonexistence is fairly common for nonlinear forms and discrete sets. The standard example is

$$S(f,c) = c_1/(1+c_2f^2) \qquad \text{for} \quad f \in \{-1,0,1\}$$

Thus the feature f can take on only one of three possible values and we choose to give S the form of the reciprocal of a quadratic polynomial. Suppose now that the best selection (of all possible forms and problems) is 1 if $f = 0$ and 0 if $f \neq 1$. Consider the case where $c_1 = 1$; we have

$$S(0,c) = 1/(1+0*f^2) = 1$$

$$S(-1,c) = S(+1,c) = 1/(1+c_2)$$

We can make $S(\pm 1,c)$ as close to zero as we want by making c_2 large; however, if we set $c_2 = \infty$, then $S(0,c)$ is ruined. The difficulty in this example is an essential one. There is no way to reparameterize $S(f,c)$ so as to obtain the best selection; yet we can come as close to it as we please.

Study of the existence question occasionally leads one to realize that the approximation form chosen must be extended in some way. A simple mathematical example of this occurs for the two-exponential form

$$S(f,c) = c_1e^{c_2f}+c_3e^{c_4f}$$

Let $c_2 = (1+\epsilon)c_4$ and expand the first term in a Taylor series after factoring out $c_1e^{c_4f}$ to obtain

$$c_1e^{c_4f}\left[1+\epsilon c_4f+\frac{(\epsilon c_4f)^2}{2!}+\frac{(\epsilon c_4f)^3}{3!}+\cdots\right]+c_3e^{c_4f}$$

This may be rewritten as

$$e^{c_4f}[c_1+c_3+c_1\epsilon c_4f+c_1(\epsilon c_4f)^2/2+\cdots]$$

Now let $c_1=-c_3$, $c_1=\alpha/\epsilon$, and then let ϵ go to zero. The result is αfe^{c_4f} and we see that this form with two exponentials also contains a function of completely different mathematical form. However, the plot of fe^f and neighboring curves in Fig. 10 shows that there is nothing exceptional about $S(f,c)$ near this curve. Even so, the coefficients are $c_1 = +\infty$, $c_3 = -\infty$, $c_2 = c_4$, with

$$(c_4-c_2)*c_1 = \alpha$$

There is a singularity in the parameterization near this curve, much as there is a singularity at the north and south poles for the geographic coordinates parameterization of the globe.

A variation of this phenomenon occurs with the piecewise forms. Consider piecewise linear forms (broken lines) with variable break points.

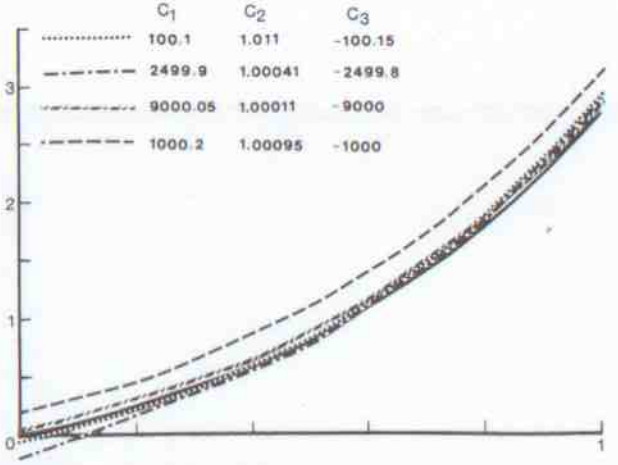


FIG. 10. The curve fe^f and nearby curves of the form $c_1e^{c_2f} + c_3e^f$ with various values of c_1, c_2, and c_3.

Figure 11 shows two things that can happen when the break points come together. In Fig. 11a we see that two of them can converge, so that the result is a step function with a jump discontinuity. In Fig. 11b we see that four break points can converge, so that an isolated peak (a "delta" function) of arbitrarily small base and large height is obtained.

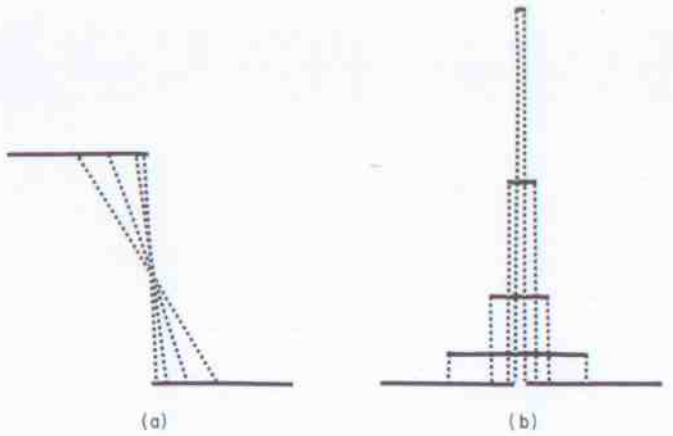


FIG. 11. Two ways that nonlinear break points in a broken line form can introduce new forms: (a) a jump discontinuity and (b) a "delta" function.

Study of the existence question can have implications for computations in the following way. If either nonexistence or the need for extending the definition are discovered, then one can expect computational difficulties. For example, if one is using the two-exponential form $c_1e^{c_2t} + c_3e^{c_4t}$ and the best approximation is fe^t (or nearly so), then the computations become extremely ill conditioned and normally collapse in a blizzard of highly magnified round-off errors.

So far we have discussed only classical mathematical forms, and we expect the same phenomena to occur for the tree and algorithm forms. A very interesting open question is whether other phenomena may occur.

6.6.2 *The Uniqueness Question*

One is usually not interested in this question *per se*; any best (or good) approximation will do. However, its study, like that of existence, can give insight into computational difficulties that may arise.

Global uniqueness is a rare property except for linear problems. This fact is intuitively illustrated by the simple problem of finding the closest point on a curve (a class of algorithms) from a given point (the optimal algorithm). This is illustrated in Fig. 12.

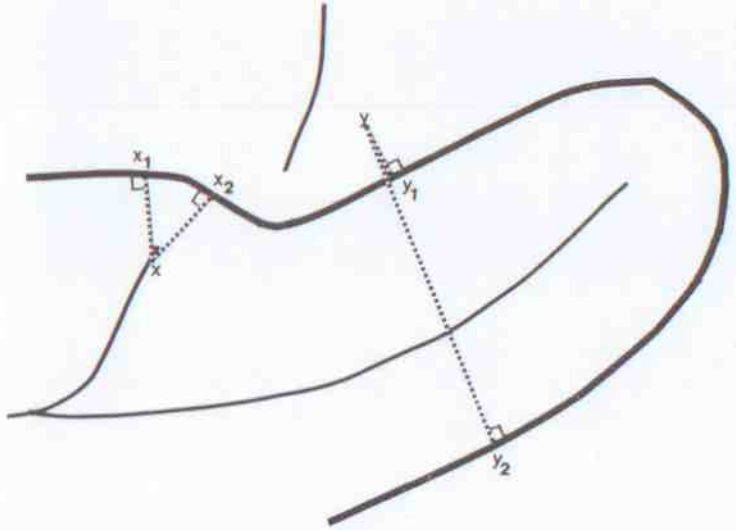


FIG. 12. Illustration of nonuniqueness of best approximation for a nonlinear problem. In a linear problem, the curve would be a straight line and every point would have a unique closest point on the line. The heavy line represents algorithms obtained by the mapping S. The light line represents problems where more than one algorithm is best.

Two other properties of the uniqueness question are illustrated by Fig. 12. First, almost all points have a unique best approximation even if a few do not. Second, we see that when there is more than one best approximation, they tend to be reasonably separated from one another. The point x, for example, has best approximations x_1 and x_2. Finally, the point y illustrates the most difficult situation where even though the closest point (y_1) is uniquely determined, there is another point (y_2) much further away, which is locally best and unique. That is to say, there is no point close to y_2 that is closer to y than y_2 is.

There are enormous computational implications of the phenomena illustrated in Fig. 12. First, and somewhat less important, one can expect trouble at those points where two or more closest points are close together. This occurs near the three ends of the "lines of nonuniqueness" in Fig. 12. More important is the fact that computational schemes are almost always local in nature and thus might well locate y_2 as the closest point to y. Further, such schemes usually give no inkling that there might be a point much closer to y. Note that this unfortunate situation occurs when we find a bad approximation (y_2 is far from y), and our limited experience in these matters does support the hope that "good" locally best approximations are likely to be global best approximations.

6.6.3 *The Characterization Question*

A characterization theorem gives some property of a best approximation, which characterizes it, i.e., which allows us to distinguish it from other approximations. An elementary approach to the question goes as follows: If we have a best approximation $S(F,C^*)$ with best coefficients C^*, then we have minimized something, namely, our measure of performance $\| p(S(F,C),F) \|$. At minima we have derivatives equal to zero. Therefore, a characteristic property comes from the equations that result from evaluating the derivative of the measure of performance and setting it equal to zero.

The application of this approach is straightforward in many instances, for example, the derivation of the normal equations for least squares approximations. In other instances, the characteristic conditions might appear to be completely unrelated to this approach. However, there usually is a direct relationship. For example, the conditions for optimality in linear programming problems is obtained this way modulo the changes necessary to include "differentiation" at the corners of multidimensional polyhedra. As an example, we derive the classical alternation theorem of minimax approximation using this elementary approach. Assume we want to ap-

proximate $f(x)$ by $S(c,t)$ with coefficients $c = c_1, c_2, \ldots, c_n$, so that

$$\max_t |f(t) - S(c,t)| = \text{minimum}$$

Then we want

$$\frac{\partial}{\partial c_j} \max_t |f(t) - S(c,t)| = 0 \qquad \text{for} \quad j = 1, 2, \ldots, n$$

Now, the maximum only occurs at the extrema of $|f - S|$ and if we denote them by t_i^*, $i = 1, 2, 3, \ldots$, we have

$$\frac{\partial}{\partial c_j} |f(t) - S(c,t)|_{\text{at } t_i^*} = 0 \qquad \text{for} \quad j = 1, 2, \ldots, n, \quad i = 1, 2, 3, \ldots$$

We now differentiate off the absolute value sign to get

$$\text{sign } |f - S|_{\text{at } t_i^*} \frac{\partial}{\partial c_j} S(c,t)_{\text{at } t_i^*} = 0$$

$$\text{for} \quad j = 1, 2, \ldots, n, \quad i = 1, 2, 3, \ldots$$

If $S(c,t)$ is linear, i.e., $S(c,t) = \sum_{j=1}^{n} c_j \phi_j(t)$, then we have

$$\text{sign } |f - S| \, \phi_j(t)_{\text{at } t_i^*} = 0 \qquad \text{for} \quad j = 1, 2, \ldots, n, \quad i = 1, 2, 3, \ldots \tag{1}$$

That this is a variation of the alternation theorem is seen as follows [for the case of polynomial approximation, $\phi_j(t) = t^{j-1}$]. First note that there must be at least n extrema t_i^* because otherwise we could find a polynomial $S(d,t)$ of degree $n-1$ so that

$$S(d,t_i^*) = \text{sign } |f(t_i^*) - S(c,t_i^*)| \qquad \text{for} \quad i = 1, 2, 3, \ldots, k \leq n \tag{2}$$

which contradicts the preceding relationship (1). More generally, the extrema t_i^* must occur with a combination of signs so that it is impossible to achieve (2) with any choice of coefficients d. Thus, using elementary properties of polynomials, one finds that there must be a set of extrema t_i^* so that

$$\text{sign } |f - S|_{\text{at } t_i^*} = (-1)^i \quad \text{or} \quad (-1)^{i+1} \qquad \text{for} \quad i = 1, 2, \ldots, n+1$$

This is the classical alternation property that characterizes best minimax approximations.

The main point made is that almost all characterization conditions come from setting derivatives equal to zero, even though in some cases it may look much different because of special situations or because the conditions have been manipulated after equating the derivatives to zero.

The implication for computation is that they also are based on finding coefficients where the derivative is zero. In many situations the key to an effective computational procedure is to find a proper interpretation of the derivative in the problem at hand. These procedures are generally iterative in nature (unless one is lucky) and share many of the computational properties of similar methods of elementary numerical analysis (e.g., Newton's method, secant method, bisection, fixed point iteration). Unfortunately, these shared properties are not that attractive in high-dimensional problems. That is, some of them are slow to converge, computationally expensive, or difficult to initialize for convergence. Some methods may have all three of these unattractive properties in certain cases.

6.7 Conclusions, Open Questions, and Problems

One objective of this section is to explore the applicability of approximation theory to the algorithm selection problem. We conclude that there is an intimate relationship here and that approximation theory forms an appropriate base upon which to develop a theory of algorithm selection methods. We also conclude that approximation theory currently lacks much of the necessary machinery for the algorithm selection problem. There is a need to develop new results for and apply known techniques to these new circumstances. The final pages of this chapter form a sort of appendix, which lists 15 specific open problems and questions in this area.

We note that there is a close relationship between the algorithm selection problem and general optimization theory. This is not surprising since the approximation problem is a special form of the optimization problem. We have not attempted to detail this relationship here, but one may refer to Rice (1970), where the relationship between nonlinear approximation and optimization is explored.

We conclude that most realistic algorithm selection problems are of moderate to high dimensionality and thus one should expect them to be quite complex. One consequence of this is that most straightforward approaches (even well-conceived ones) are likely to lead to enormous computations for the best selection. Indeed, the results of Rabin (1974) suggest that this complexity precludes the determination of the best selection in many important cases.

Finally, we reiterate the observation that the single most important part of the solution of a selection problem is the appropriate choice of the form for the selection mapping. It is here that theories give the least guidance and that the art of problem solving is most crucial.

We list 15 questions that are given or suggested by the developments of this chapter.

(1) What is the relationship between tree forms and piecewise linear forms? Can all tree forms be made equivalent to some piecewise form, linear or nonlinear?

(2) What are the algorithm forms for the standard mathematical forms? Do they suggest useful simple classes of algorithm forms? See Hart *et al.* (1968, Chapter 4) for algorithm forms for some polynomial and rational forms.

(3) Determine specific classes of tree forms where the current machinery of nonlinear approximation is applicable.

(4) Develop some general approaches (or methods) to classifying problems within a problem space. This is related to the next problem.

(5) Develop an abstract machinery for analyzing optimal features. Such a machinery might well combine the theoretical ideas of n-widths and/or entropy (see Lorentz, 1966) with the intuitive ideas of performance profiles (see Lyness and Kaganove, 1976).

(6) What is the nature of the dependence of the degree of convergence on the dimensionality of the problem? Some results are known for polynomial approximation to multivariate functions. Are these typical of what one should expect in general?

(7) What is the nature of the dependence of complexity on the dimensionality of the problem? Can results of (6) be translated directly into statements about complexity?

(8) Obtain more precise information about the nature of real world functions. The generalities used in this report were obtained by selecting a large number of empirically determined functions from the "Handbook of Chemistry and Physics" (1960) and then observing how effective polynomial approximation is. Are the results of this experiment representative of other contexts? Can more precise information about the properties of such classes be obtained?

(9) Determine the computational complexity of the following specific problems. For simplicity, one may use one evaluation of $f(x)$ as the unit of computation and ignore all other work.

(a) Approximation to $f(x)$ via interpolation by polynomials. Assume various kinds of smoothness for $f(x)$.

(b) Least squares approximation to $f(x)$ on $[0,1]$ by polynomials. Assume various kinds of smoothness for $f(x)$.

(c) Evaluation of $\int_0^1 f(x)\,dx$. This is closely related to the least squares problem.

(10) Formulate a more precise and general concept of robustness.

(11) Develop useful mechanisms to embed certain classes of discrete forms into continuous ones. This is particularly relevant for nonstandard mathematical forms.

(12) Develop techniques to partition high-dimensional problem sets into subsets where good linear approximations are possible. A particular instance would be to develop adaptive algorithms for piecewise linear (no continuity) approximations in high dimensions. See Pavlidis (1973) for some work in one dimension.

(13) Develop existence theorems for various classes of tree form approximations. Do the difficulties of coalesced knots that occur in spline approximation have an analogy in general tree forms?

(14) What are the relationships between best algorithm selection and the results in automata theory about computability and computational complexity?

(15) Is there any way to "differentiate" the tree form so as to obtain a local characterization theorem?

References

Abell, V. (1973). "Queue Priorities in the Purdue MACE Operating Systems," PUCC Publ. ZO QP-1. Purdue University, West Lafayette, Ind.

Blue, J. L. (1975). "Automatic Numerical Quadrature—DQUAD," Comput. Sci. Tech. Rep. No. 25. Bell Telephone Labs, Murray Hill, N.J.

Casaletto, J., Picket, M., and Rice, J. (1969). A comparison of some numerical integration programs. *SIGNUM Newslett.* **4**, 30–40.

Coffman, E. G., and Denning, P. J. (1974). "Operating Systems Theory," Chapters 3 and 4. Prentice-Hall, Englewood Cliffs, New Jersey.

de Boor, C. (1971). CADRE—an algorithm for numerical quadrature. *In* "Mathematical Software" (J. Rice, ed.), pp. 417–449. Academic Press, New York.

Einarsson, B. (1974). Testing and evaluation of some subroutines for numerical quadrature. *In* "Software for Numerical Mathematics" (D. J. Evans, ed.), pp. 149–157. Academic Press, New York.

Gentleman, M. W. (1972). Algorithm 424, Clenshaw-Curtis quadrature. *Commun. ACM* **14**, 337–342 and 353–355.

"Handbook of Chemistry and Physics." (1960). Handbook Publishers Inc., Sandusky, Ohio.

Hart, J. F. *et al.* (1968). "Computer Approximations." Wiley, New York.

Kahaner, D. K. (1971). Comparison of numerical quadrature formulas. *In* "Mathematical Software" (J. Rice, ed.), pp. 229–259. Academic Press, New York.

Krogh, F. T., and Snyder, M. V. (1975). "Preliminary Results with a New Quadrature Subroutine," Comput. Memo. No. 363, Sect. 914. Jet Propulsion Laboratory, Pasadena, Calif.

Lorentz, G. G. (1966). "Approximation of Functions." Holt, New York.

Lyness, J. N., and Kaganove, J. J. (1976). Comments on the nature of automatic quadrature routines. *ACM Trans. Math. Software* **2**, 65–81.

Patterson, T. N. L. (1973). Algorithm 468, algorithm for automatic numerical integration over a finite interval. *Commun. ACM* **16**, 694–699.

Pavlidis, T. (1973). Waveform segmentation through functional approximation. *IEEE Trans. Comput.* **c-22**, 689–697.

Piessens, R. (1973a). An algorithm for automatic integration. *Angew. Inf.* **9**, 399–401.

Piessens, R. (1973b). "A Quadrature Routine with Round-off Error Guard," Rep. TW17. Appl. Math. Prog. Div., Katholicke Universiteit, Leuven.
Rabin, M. O. (1974). Theoretical impediments to artificial intelligence. *Proc. IFIP, 1974* pp. 615–619.
Rice, J. R. (1970). Minimization and techniques in nonlinear approximation. *Stud. Numer. Anal.* **2**, 80–98.
Rice, J. R. (1975). A metalgorithm for adaptive quadrature. *J. Assoc. Comput. Mach.* **22**, 61–82.
Wilkes, M. (1973). Dynamics of paging. *Comput. J.* **16**, 4–9.

Parallel Processing of Ordinary Programs*

DAVID J. KUCK

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois

1. Introduction

In the decade 1965–75 we have seen a number of changes in computer technology and computer use. Integrated circuits have arrived and with

* This work was supported in part by the National Science Foundation under Grant No. US NSF DCR73-07980 A02.