Purdue University
Purdue e-Pubs

Department of Computer Science Technical Reports | Department of Computer Science

1974

The Algorithm Selection Problem II—Two Concrete Problems: Numerical Analysis Quadrature—Algorithms, Operating Systems—Scheduling Design

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
74-117

Rice, John R., "The Algorithm Selection Problem II—Two Concrete Problems: Numerical Analysis Quadrature—Algorithms, Operating Systems—Scheduling Design" (1974). *Department of Computer Science Technical Reports.* Paper 69.
https://docs.lib.purdue.edu/cstech/69

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

THE ALGORITHM SELECTION PROBLEM II - TWO CONCRETE PROBLEMS
Numerical Analysis - Quadrature Algorithms
Operating Systems - Scheduling Algorithms

John R. Rice
May, 1974

CSD-TR 117

This work was supported in part by NSF Grant GP-32940X.

1. INTRODUCTION. The purpose of this report is to examine concrete problems within the framework of the abstract models developed in [10]. Our main objective here is to examine these two problems and not to solve them. We do, however, give an outline of how a realistic selection might proceed. The next report considers a game playing algorithm (Tic-Tac-Toe) and examines the relationship of "learning" (in the Artificial Intelligence sense) to the algorithm selection problem. These three concrete examples are selected to show the diversity of real problems that fit into the abstract models of [10]. 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 are:

A. The selection of a numerical quadrature algorithm;

B. The selection of a scheduling algorithm for an operating system.

They 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 dimensionally.
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 has not been collected systematically and is 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.

The game playing algorithm selection problem considered in the next report is quite different. It is much less complicated (a perfect algorithm is already known), it involves unfamiliar mathematical functions (that is, unfamiliar for approximation and optimization), the performance criterion is simple. As with the scheduling problem, it does involve a dynamic process but the consequences are less formidable since the problem is so much less complicated.

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 a simpler, concrete case;
Formulation of a specific and simpler selection problem;
Discussion of the simpler problem and the computations required to solve it.

2. THE SELECTION OF QUADRATURE ALGORITHMS. The general case of this problem may be expressed in one of the two following ways:

A. Given a collection of functions (with reasonabley well known attributes), which one of the 15 to 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 to 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 in terms of the known information.

This general problem is modelled by the situation in Section 5 of [10] which involves spaces for the problems, the features, the criteria, the algorithms and the performance measures. These spaces are described as follows:

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.

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. These 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 functions 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 which have been overlooked in this list.

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) which 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 [11]). The actual algorithms one might consider are mentioned in the various references and many of them are named later.

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.

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 tractible to use or describe to users.

2.1 Summary of Experimental Testing Results. 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, Pickett and Rice [2]: A set of 50 functions;

B. Kahaner [6]: A set of 21 functions;

C. de Boor [4], Lyness: 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 six 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 detailed references are not given here, one must refer to the test reports.

1. Casaletto, Pickett and Rice [2]. Complete details not reported.
Test set A with $\varepsilon = 10^{-1}, 10^{-2}, \ldots, 10^{-8}$

Algorithms: QUAD - Adaptive Simpson Rule
QUADS4 - Adaptive 4-point Games Quadrature
QUADS6 - Adaptive 6-point Games 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 [6]. Extensive tables of detailed results.
Test set B with $\varepsilon = 10^{-3}, 10^{-6}, 10^{-9}$

Algorithms: 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
SIMPSN - Adaptive Simpson rule
SQUANK - Improved version of SIMPSN

3. de Boor [4]. Results compatible with Kahaner plus graphs.
Test set B with $\varepsilon = 10^{-3}, 10^{-6}, 10^{-9}$ plus test set C.

Algorithm: CADRE - Adaptive Romberg with cautious extrapolation.

4. Gentleman [5]. Considerable detail.
Test set A with $\varepsilon = 10^{-1}, 10^{-2}, \ldots, 10^{-8}$

Algorithm: CCQUAD - Curtis-Clenshaw quadrature.

5. Patterson [7]. Partial results reported involving CADRE and QSUBA.
Test set selected from A and B plus three others; total of 13 functions.

Algorithms: CADRE - Adaptive Romberg with cautious extrapolation

QSUB - Iterated Gauss-Kronrod rules up to 255 points
QSUBA - Adaptive version of QSUB
SQUANK - Improved adaptive Simpson rule

6. Piessens [8]. Complete details not reported
Test set A with $\epsilon = 10^{-2}, 10^{-3}, \ldots, 10^{-13}$
Algorithms: AIND - Adaptive Gauss-Kronrod rules up to 21 points
CCQUAD - Curtis-Clenshaw quadrature
HRVINT - Improved version of HAVIE (Adaptive Romberg)
SQUANK - Improved version of SIMPSN (Adaptive Simpson)

7. Piessens [9]. Considerable detail given, some round-off effects studied.
Test set A with $\epsilon = 10^{-5}, 10^{-7}$ (with noise), 0
Algorithms: AIND - Adaptive Gauss-Kronrod rules up to 21 points
CADRE - Adaptive Romberg with cautious extrapolation
SQUANK - Improved adaptive Simpson rules

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 as follows:

Suggested Improved Experiment. We assume the quadrature problem is

$$\int_0^1 h(t)dt$$

We choose a feature space with 4 dimensions:

Feature Name	Values Assumed	Remarks
Smoothness	0,1/2,1	0 is smooth, 1 is not
Singularity	[-1,2]	value is exponent of singularity
Peak	[0,100]	strength = $\frac{\text{Average size of } h(t)}{\text{(Peak base)*(Ave. size of peak)}}$
Oscillation	[0,100]	maximum frequency of oscillation

We choose four 1-parameter families of functions that represent each of the features (the performance profiles of Lyness) and then each coordinate

axis of $\mathcal{F}$ 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. $|t^2-.25|^\alpha$ has a singularity, $\sin[N(t^2+1)]$ is oscillatory and both $|t^2-.25|^\alpha + \sin[N(t^2+1)]$ and $|t^2-.25|^\alpha \sin[N(t^2+1)]$ are oscillatory with a singularity). This process gives a test set which produces a grid over the entire feature space. This test set can be combined with accuracy values of $\varepsilon = 10^{-2}, 10^{-4}, 10^{-8}, 10^{-12}$ 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 distribution in the problem space which 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) and thus one would not expect SQUANK to compare favorably with some other algorithms on the basis of this experiment.

2.2 Algorithm Selection Based on Current Information. While there is a clear need for better data on the selection problem for quadrature algorithms, it is reasonable to ask: Can one use the currently known information to develop better information about algorithm selection and performance? This question is of great interest in itself but it is also very relevant as a realistic example. The difficulty, of course, is that the current information is fragmentary, based on differing assumptions and generally incompatible in various ways. We will formulate a simplified problem where these difficulties are handled in ways that may be effective in more realistic situations.

The information on algorithm performance is fixed and we consider a subset of the algorithms, namely:

AIND, CADRE, CCQUAD, Q96, QABS, QSUBA, QUAD, QUADS6, SIMPSN, SQUANK

The features of all the test problems are to be measured. We consider two criteria of performance: efficiency and reliability. These two variables are scaled to the interval [0,1] as follows:

Efficiency: Let N_0 be the minimum number of integrand evaluations required to solve the problem (this must be estimated for each problem) and N_a be the actual number used by a particular algorithm A. Then the value of the efficiency is $N_0/N_a = p_1(A,x)$.

Reliability: Let ε_0 be the requested accuracy and ε_a be the accuracy actually achieved. The value of reliability is then taken to be

$$\varepsilon_a > \varepsilon_0: \quad p_2(A,x) = 1 - (1 - \varepsilon_0/\varepsilon_a)^2$$

$$\varepsilon_a < \varepsilon_0: \quad p_2(A,x) = 1/(1 + .1(\log \varepsilon_a/\varepsilon_0)^2)$$

This places a severe penalty on failing to achieve ε_0, and a mild penalty on achieving much more accuracy than ε_0. These conventions allow us to find the performance vector $(p_1(A,x),p_2(A,x))$ and we introduce a criteria unit vector (w_1,w_2) and the norm of $p(A,x)$ is then

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

Thus we are able to compute (or estimate) the performance for all the tests mentioned above. We now choose to model the performance behavior of the algorithms by a linear mapping:

$$p_1(A,x) \sim \alpha_0(A) + \sum_{i=1}^{4} \alpha_i(A)f_i(x) = \rho_1(A,x)$$

$$p_2(A,x) \sim \beta_0(A) + \sum_{i=1}^{4} \beta_i(A)f_i(x) = \rho_2(A,x)$$

Recall that $f_i(x)$ is the ith feature of the problem x. There is no reason to believe that this linear model is a good choice, but we want to keep this aspect of the problem simple. The selection mapping is then defined as follows:

Selection Mapping: Given x and w, compute the predicted performance for each algorithm and select the algorithm with maximum performance. Thus, the algorithm A^* selected satisfies

$$w_1\rho_1(A^*,x) + w_2\rho_2(A^*,x) = \max_A [w_1\rho_1(A,x) + w_2\rho_2(A,x)]$$

Recall the discussion of the various norms that one can use in optimizing the selection and the remarks that the exact choice was usually not crucial (say compared to the choice of the selection mapping's form). In the present situation we have chosen a mediocre (at best) form for the selection mapping and we have considerable uncertainty in the data for this problem. Thus we propose to choose a norm that is most convenient for computation. In fact we propose to use least squares approximation for the problems

$$p_1(A,x) \sim \rho_1(A,x)$$

$$p_2(A,x) \sim \rho_2(A,x)$$

and not determine directly what this implies about the norm used in the choice of selection mapping.

We have then 20 linear least squares problems (two for each of 10 algorithms) with 5 unknown coefficients and perhaps 10 to 100 data points distributed unevenly in the four dimensional space $\mathcal{F}$. The straight forward approach is to determine the $\alpha_i(A)$ so as to minimize

$$\sum_{k=1}^{K} [p_1(A,x_k) - (\alpha_0(A) + \sum_{i=1}^{4} \alpha_i(A)f_i(x_k))]^2$$

where K is the number of data points x_k for the algorithm A. This is notorious for giving poor results when the data is very uneven (as it is here) and the terms in the sum of squares should be weighted by the volume in that each data point represents. This corresponds to using a crude Riemann sum to approximate the 4-dimensional integral

$$\int_{\mathcal{F}} [p_1(A,x) - (\alpha_0(A) + \sum_{i=1}^{4} \alpha_i(A)f_i(x))]^2 dx$$

The problem of associating volumes with the data points is non-trivial, but feasible. A viable alternative is to choose a fixed set of points and determine a function value for f(x) by a weighted average of nearby points. The exact details of the least squares norm to be used is not germane to the main point here, so we leave it aside.

We reach the conclusion that it is practical to take the currently available experiemental data and determine an algorithm selection procedure which has real hope of being good. Indeed, if the linear mapping is replaced by a simple combination of splines of degree 0 or 1 (i.e. use broken lines to estimate $p_1(A,x)$ and $p_2(A,x)$) then one should expect to obtain a useful selection procedure.

3. THE SELECTION OF OPERATING SYSTEMS SCHEDULING ALGORITHMS. The general case of this problem may be experessed 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 paper. We will formulate this problem more precisely within the framework provided by the abstract models of [10]. 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.

This selection problem, like the preceding one involving quadratures, is modeled as in Section 5 of [10]. This model involves spaces for the problems, the features, the criteria, the algorithms and the performance measures. THese spaces are described as follows:

Problem Space: This space consists of configurations of computer runs which are mixtures of batch, remote batch, timeshared 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.

Feature Space: The features of a configuration of computer runs are a combination of the features of the individual jobs. The features of indi-

vidual 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 likelyhood of page requests or job roll-outs

In addition features of the total problem configuration should be considered such 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 which are not included in this list.

Algorithm Space. A fair variety of scheduling algorithms have been proposed and analyzed to a certain extent [3], [12]. 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.

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 are very realistic, but efforts to measure the progress made toward satisfying them usually involve thruput 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

Thruput - total number of jobs processed per unit time, number of CPU hours billed per day , etc.

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 one) is simple, but often satisfactory.

3.1 Outline of an actual scheduling algorithm. We present a version of the scheduling algorithm used on the CDC 6500 system at Purdue University [1]. 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 $\vec{r} = (r_1, r_2, r_3, r_4, r_5, r_6)$ 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

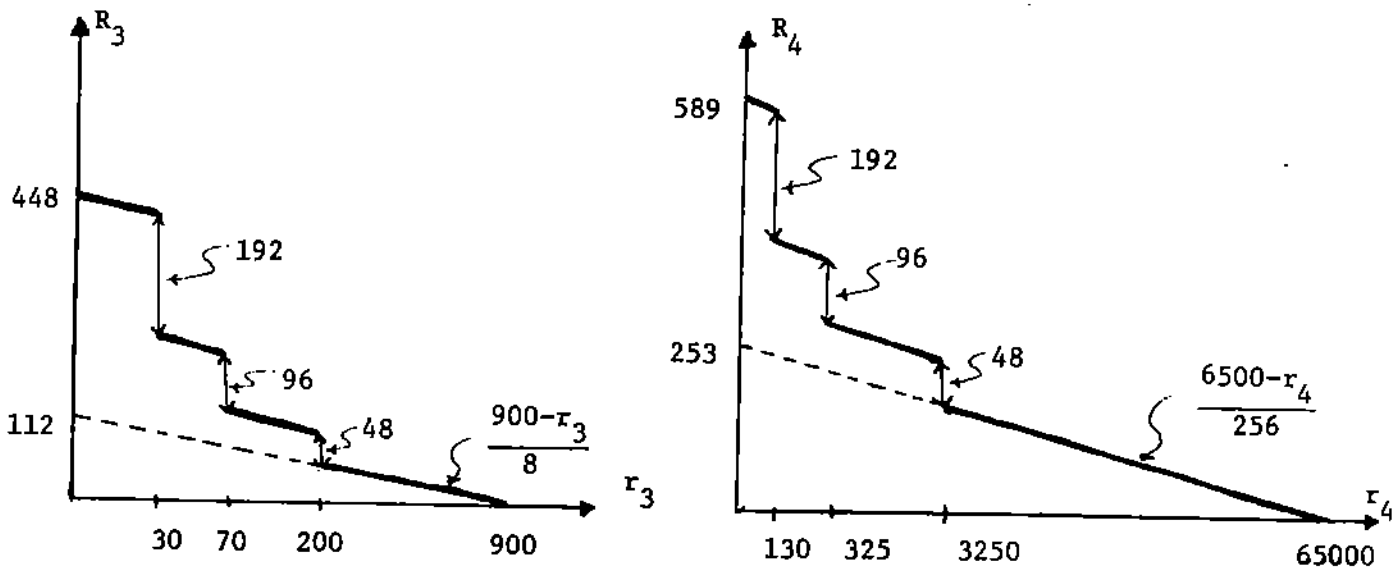
$$R_1(r_1) = 2^6 * r_1$$

$$R_2(r_2) = |r_2 - 150100|/128$$

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

$$R_6(r_6) = r_6$$

and R_3 and R_4 are shown in Figure 1 (a) and (b).



(a) priority contribution for CPU time (b) priority contribution for I/O units

Figure 1. Graphs of the functions $R_3(r_3)$ and $R_4(r_4)$. The horizontal axes are not drawn to scale. Each function is linear plus three step functions.

This function QP involves about 22 coefficients.

3.2 A Simplified Scheduling Algorithm Selection Problem. The algorithms considered involve 3 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)

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

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

where

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

$$R_2(r_2) = a_2 * (150100 - r_2)$$

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

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

$$R_5(r_5) = a_{15} * |r_5 - a_{16}|_+^0 + a_{17} * |a_{18} - r_5|_+$$

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

Recall the notation

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

This queue priority function is a slightly modified and simplified version of the one in the previous section.

We choose a three dimensional performance measure space with

$\vec{p} = (p_1, p_2, p_3)$ where

p_1 = (Mean internal processing time for short jobs)/1000

p_2 = (Mean internal processing time for other jobs)/4000

p_3 = (Mean PROCSY response time for standard short tasks)/10

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

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

where w is from the three dimensional criteria space with $w_1 \geq 0$ 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 w
2. The operating system measures the configuration features f_1, f_2, f_3
3. The appropriate best coefficients $\vec{a}$ are used for these values of w_i and f_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 to 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.

3.3 An Approach to the Selection of a "Best" Scheduling Algorithm. 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. $OPT(\vec{w},\vec{f})$ 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 f_{jk}, $k = 1$ to m_f and execute the algorithm

$$\underline{\text{For}}\ \ell = 1\ \underline{\text{to}}\ m_w,\ k = 1\ \underline{\text{to}}\ m_f\ \underline{\text{do}}\ a_i(\ell,k) = \text{OPT}(\vec{w}_\ell, \vec{f}_k)$$

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 to 200 evaluations of functions (that is, $\vec{p}$ as a function of $\vec{a}$) 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 $\vec{a}$ is inadequate. This is not likely since the form is the one in current use.
3. The linear form for the $\vec{a}$ 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 $||\vec{p}||$ 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 $\vec{f}$ (due to the changing job configuration) and the values of $\vec{a}$ (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 availalbe. 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 $||\vec{p}||$ which are fairly tightly associated with values of $\vec{w}$, $\vec{f}$ and $\vec{a}$. This means that it is, in principle, feasible to carry out the optimization problem. A simplified example of the situation is shown in Figure 2 where we assume there is 1 variable for $\vec{a}$, and 1 variable for $\vec{w}$ and $\vec{f}$.

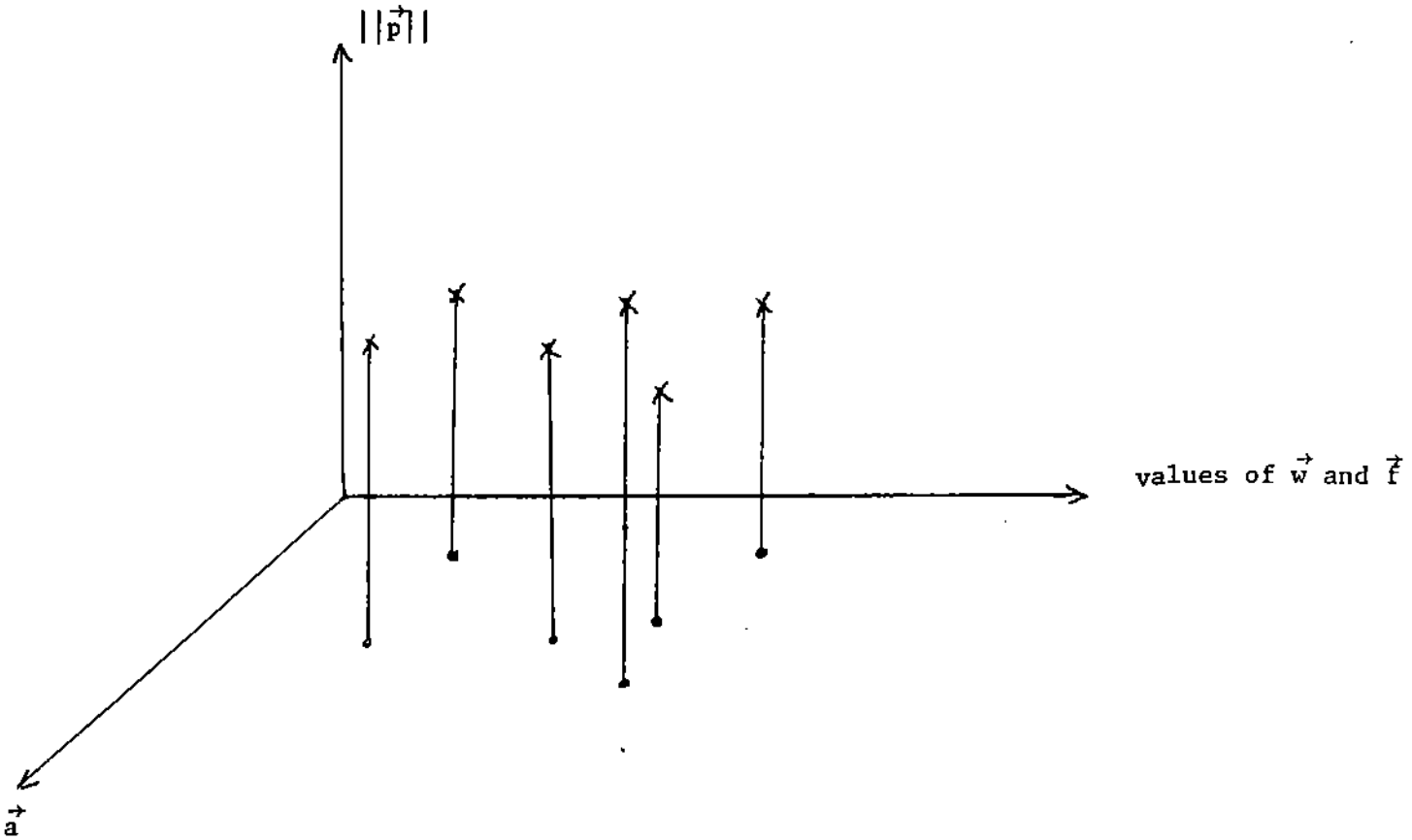


Figure 2. Function values of $||\vec{p}||$ obtained when there is no direct control over some of the arguments ($\vec{f}$ in this case).

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. While such rules are not standard, they should not be exceptionally difficult to obtain and only rough accuracy is required here. Note that certain values of $\vec{f}$ might be very uncommon and hence the optimization obtained there might be unreliable. Fortunately, the rarity of these values of $\vec{f}$ 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 $||\vec{p}||$ and all other quantities within a 15 minute time interval. This corresponds to 1 function evaluation. Thus we are led to assume that one evaluation of OPT takes from 1/2 to 2 days of system time. The inefficiency due to the lack of control over setting parameters will probably double this time, say to 3 days. The number of evaluations of OPT needed to obtain semi-reasonable reliability in the α_{ij} computations is probably the order of 50 or 100. This implies about 6 months or a year to select the best scheduling algorithm.

Note that this approach is much different than the common theoretical approach. There one assumes some model for the computer operation and then analytically obtains a good (or optimum) scheduling algorithm for this 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, of course, yet to be seen just how feasible or effective this direct approach will be.

REFERENCES

[1] Abell, V., Queue Priorities in the Purdue MACE Operating Systems. PUCC Publication ZO QP-1, Dec., 1973.

[2] Casaletto, J., Picket, M. and Rice, J., A Comparison of some Numerical integration programs. SIGNUM Newsletter, 4 (1969), 30-40.

[3] Coffman, E.G. and Denning, P.J., Operating Systems Theory, Prentice Hall, Englewood Cliffs, N.J., (Chapters 3 and 4).

[4] de Boor, C., CADRE - An algorithm for numerical quadrature, in Mathematical Software (Ed. J. Rice), Academic Press, New York, (1971) 417-449.

[5] Gentleman, M.W., Algorithm 424, Clenshaw-Curtis quadrature, Comm. ACM, 14 (1972) 337-342 and 353-355.

[6] Kahaner, D.K., Comparison of Numerical Quadrature Formulas, in Mathematical Software (Ed. J. Rice), Academic Press, New York (1971) 229-259.

[7] Patterson, T.N.L., Algorithm 468, Algorithm for Automatic Numerical Integration Over a Finite Interval, Comm. ACM, 16 (1973) 694-699.

[8] Piessens, R., An Algorithm for Automatic Integration. Angewandte Info. (1973) 399-401.

[9] ————————, A Quadrature Routine with Round-off Error Guard. Report TW 17, Appl. Math. Prog. Div., Katholicke Universiteit Leuven, March 1973, 11 pages.

[10] Rice, J.R., The Algorithm Selection Problem-Abstract Models. CSD-TR 116, Purdue University, May (1974) 21 pages.

[11] ————————, A Metalgorithm for Adaptive Quadrature, J. Assoc. Comp. Mach., to appear.

[12] Wilkes, M., Dynamics of Paging. Computer J., 16 (1973) 4-9.