A Learning Machine for Job Sequencing in a General-purpose Computer System

Richard Jon Taylor
University of Central Florida

A LEARNING MACHINE FOR JOB SEQUENCING IN A GENERAL-PURPOSE COMPUTER SYSTEM

BY

RICHARD JON TAYLOR
B.S., State University of New York at Albany, 1968

RESEARCH REPORT

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Graduate Studies Program of
Florida Technological University, 1973

Orlando, Florida

TABLE OF CONTENTS

INTRODUCTION 1

I. BACKGROUND 3

II. GENERAL MODEL STRUCTURE 4

III. GENERAL MODEL OPERATION 6

IV. DETAILED DISCUSSION OF MODEL STRUCTURE AND OPERATION 7

Job Queue Model
Resource Allocation Model
Processor Model
Adaptive Model

V. MODEL RATIONALE AND INTERPRETATION 19

LIST OF REFERENCES 24

LIST OF TABLES

Table		Page
1.	Possible AM inputs on completion of job i	17

LIST OF ILLUSTRATIONS

Figure | Page
1. Block diagram of system model 5
2. Job class definition 8
3. Example of divisions of processor storage 10
4. Distribution of actual run time for job i 11
5. Matrix of decision|input conditional probabilities 16

INTRODUCTION

"It is neither impossible nor unreasonable to build computers which are at least 1000-fold and perhaps 10,000-fold or more, faster than the best machines in design today."[1]

"...a search of all the paths through the game of checkers involves some 10^{40} move choices, in chess, some 10^{120}. If we organized all the particles in our galaxy into some kind of parallel computer operating at the frequency of hard cosmic rays, the latter computation would still take impossibly long; we cannot expect improvements in hardware alone to solve all our problems !"[2]

Until the "supercomputers" become available for widespread use, we may not know with any certainty whether speed alone will solve some of our problems in maintaining consistently high computer system "throughput" and utilization.

Meanwhile, extensive work is being conducted in the area of developing machines that exhibit what could be considered to be intelligent behavior.

We will attempt in this report to show how some of the principles of artificial intelligence might be applied to computer system resource allocation in order to improve system performance.

---

[1]Willis H. Ware, Limits in Computing Power, Rand Corp. Paper P-4710 (Santa Monica, Calif: Rand Corp., 1971), p. 18.

[2]Marvin Minsky, Computers and Thought, ed. by Edward A. Feigenbaum and Julian Feldman (New York: McGraw-Hill Book Company, 1963), p. 408.

The general type of computer system represented by the model is a multiprogramming system that can execute several jobs in its main storage concurrently.

We will attempt to avoid some of the complexities of modeling computer systems and focus on the resource allocation and artificial intelligence aspects of the model.

I. BACKGROUND

Modeling of computer systems and performance evaluation has become more difficult over the past several years due to the increased complexity of the large machines now available. However, some of the general approaches have been studied. Work based on automata theory and theory of algorithms has been supported [3] while others have considered actual system simulation problems [4].

Several surveys of the years from the mid-1950's to 1971 are available that discuss developments and problems in artificial intelligence [2,5,6] and the Soviet literature has also been reviewed [7].

Much of the basic theory in pattern recognition and artificial intelligence is covered by Nilsson [8,9].

II. GENERAL MODEL STRUCTURE

The system model is made up of four components: (1) a job queue model, referred to as the JQM, (2) a resource allocation model, referred to as the RAM, (3) a processor model, referred to as the PM, (4) an adaptive model, referred to as the AM.

The job queue model is the source of jobs which the system is to process. Specified numbers of jobs are generated at various times during system operation. Processor storage requirement and estimated run time are the defining parameters of each job. The specification of these parameters in the JQM is governed by a predefined statistical distribution.

Allocation of processor storage is performed by the RAM, or resource allocation model. The RAM examines jobs generated by the JQM and selects those to be sent to the processor model. The alternative allocation algorithms in the RAM contain parameters which can be adjusted by the adaptive model to improve system performance.

The processor model, or PM, simulates execution of the jobs sent to it by the RAM. Execution of a job by the PM is represented by processor storage utilization and variation in run time due to contention for system resources.

Learning and adaptive algorithms in the AM enable it to decide how it should alter the resource allocation model in order to improve system performance according to a specified criterion.

A block diagram of the system model with its four components is shown in Figure 1.

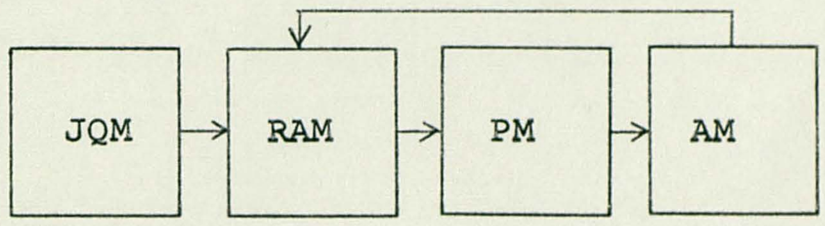

Fig. 1--Block diagram of system model

III. GENERAL MODEL OPERATION

Time sequencing of events in the system model is based on the operation of the system for a specified number of hours.

At the beginning of each hour the JQM generates a group of jobs to be placed in the input queue. At the end of each hour the adaptive model examines system performance and alters the RAM if this performance is unacceptable.

Each minute of each hour the RAM examines the jobs in the input queue and based on information concerning available processor storage it selects jobs that are sent to the PM. Also at each minute the PM accepts jobs from the RAM, alters the accepted jobs' run times according to a statistical distribution, releases any job whose execution is complete and updates the state of currently available processor storage.

IV. DETAILED DISCUSSION OF MODEL STRUCTURE AND OPERATION

Job Queue Model

The job queue model, or JQM, generates jobs that make up the input to the system. At the beginning of each hour of model operation a specified number of jobs is placed in the input queue. That is, at hour k, m_k jobs are produced by the JQM and made available to the RAM for possible execution in the PM.

A job i is represented by the parameters c_i and r_i, where c_i is the amount of processor storage required to execute the job and r_i is the estimated time the job will actually occupy the system processor.

Particular values of c_i and r_i for each job generated by the JQM are determined by predefined job class and statistical distributions. Job classes A, B, C, D, E and F are defined by storage and run time limits as shown in Figure 2. The average proportion of all jobs to be selected from each class, represented by the quantities P_A, P_B, P_C, P_D, P_E and P_F, is specified. Proportions for the six classes must sum to one. Within a particular class, the c_i and r_i for a job i chosen from that class are selected at random from a uniform distribution over the acceptable storage and estimated run time ranges for the class.

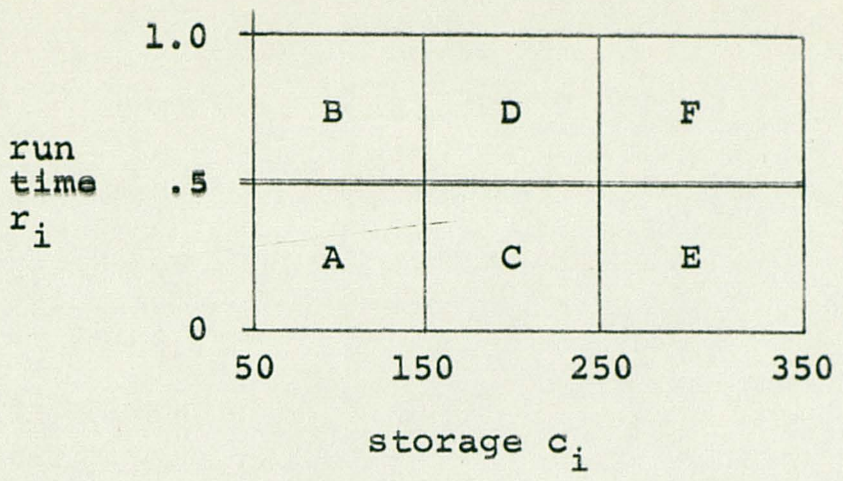

Fig. 2--Job class definition

All jobs in the input queue are eligible for selection by the resource allocation model for execution. The JQM does not assign priorities or queue position to the jobs it generates.

Resource Allocation Model

The RAM selects jobs from the input queue to be entered into the system processor.

At each minute of model operation between zero and n jobs, where n is the number of jobs in the input queue, are sent to the processor model. The decision on which jobs are to be sent is based on available processor storage and a resource allocation algorithm.

The mathematical model of allocation is in the form of an integer programming maximization problem:

$$\max_{x_i} \; w_1 \sum_{i=1}^{n} (c_i/350) r_i x_i + w_2 \sum_{i=1}^{n} (-r_i x_i) + w_3 \sum_{i=1}^{n} (-c_i/350) x_i$$

subject to $\sum_{i=1}^{n} (c_i/350)x_i \leq (c_a/350)$

$$x_i = 0 \text{ or } 1,\ i=1,2,\ldots,n.$$

After the program is solved, the variable x_i is zero if job i is not to be run and one if the job is to be sent to the PM for processing. Values of r_i are between 0 and 1; c_i is normalized by division by 350, the largest storage requirement possible for a job. The constraint c_a, also normalized, is the storage currently available in the processor as provided by the PM.

The objective function of the allocation program is made up of three sets of terms. Each set represents an approach toward allocation of processor storage. The summation $\sum_{i=1}^{n} (c_i/350)r_i x_i$, when maximized with respect to the x_i, causes jobs with the largest $c_i r_i$ values to be selected for processing. Similarly, maximizing $\sum_{i=1}^{n} (-r_i x_i)$ over the x_i results in selection of jobs with the shortest estimated run times to be sent to the PM. When the third component of the objective $\sum_{i=1}^{n} (-c_i/350)x_i$ is maximized with respect to the x_i variables, jobs with the smallest storage requirements are selected for execution.

The weights w_1, w_2 and w_3 which multiply the objective function components are nonnegative real numbers that allow adjustment of the allocation algorithm by the adaptive model.

When a job is selected by the RAM for processing, the time d_i that the job spent in the input queue is saved for later use by the AM.

Processor Model

The processor model, or PM, receives jobs from the RAM and simulates the utilization of storage by the jobs for a particular amount of run time.

At each minute of model operation the PM accepts jobs from the RAM, alters run time for new jobs based on a statistical distribution, deletes jobs that have completed their run time in the processor and updates the state of currently available processor storage c_a. A predefined maximum value of c_a, $c_{a_{max}}$, is specified and represents the size of the system processor.

If three jobs with storage requirements c_1, c_2 and c_3 are in the processor the storage divisions can be represented by the diagram in Figure 3.

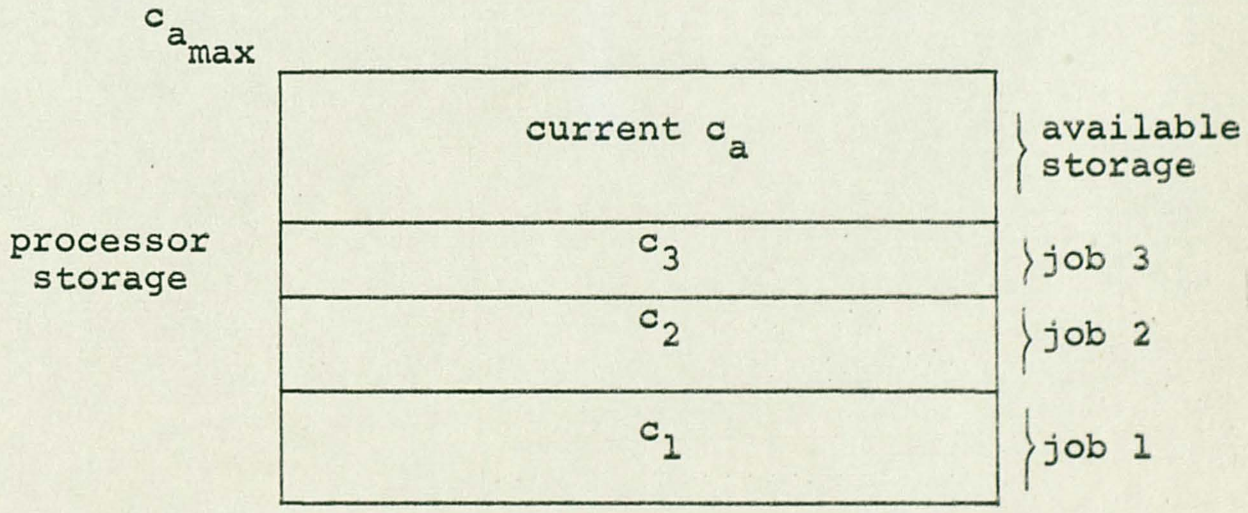

Fig. 3--Example of divisions of processor storage

Note that c_a, the available storage in the processor at a particular time, must be available for use by the resource allocation model for its storage constraint.

The estimated run time r_i can vary due to contention for system resources during processing in a computer system. To represent this variation, the PM chooses an actual run time at random from a statistical distribution with mean r_i for job i. Figure 4 shows a normal (Gaussian) distribution of the random variable p from which the actual run time p_i will be selected for job i.

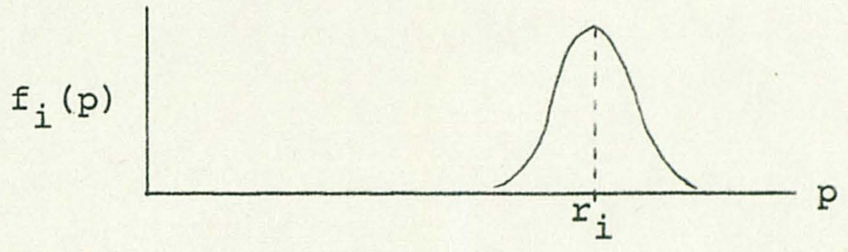

Fig. 4--Distribution of actual run time for job i

The dispersion of the variable p about the mean must be specified for the random selection process. In order to prevent extremely small or large values of p_i from being chosen from the distribution defined by $f_i(p)$, the variance must be such that only a negligible probability exists of the p_i being, say, less than $0.5r_i$ or greater than $1.5r_i$.

For a random variable p that is normally distributed with mean r_i and variance σ^2, we know that

$$P(r_i - k\sigma \leq p \leq r_i + k\sigma) = 2\Phi(k) - 1,$$

where $\Phi(k) = (2\Pi)^{-\frac{1}{2}} \int_{-\infty}^{k} \exp(-\frac{1}{2}p^2)\,dp.$[1]

[1]Paul L. Meyer, <u>Introductory Probability and Statistical Applications</u> (2d ed.; Reading, Massachusetts: Addison-Wesley Publishing Company, 1970), pp. 186-187.

In order to avoid selecting values of p_i that differ from the mean r_i by more than $0.5r_i$, we must specify the variance σ^2 that will result from the expression above when k is chosen such that $2\Phi(k)-1$ is close to one, that is, when k is such that the probability of $p_i<0.5r_i$ or $p_i>1.5r_i$ is negligible.

When $k=3.07$, $2\Phi(k)-1\simeq0.9998$, thus we must have $k\sigma=3.07\sigma\leq0.5r_i$ so we choose $\sigma=0.5r_i/3.07\simeq0.163r_i$.

Therefore, to determine the actual run time of a job i with estimated run time r_i, we select p_i at random from a normal distribution with mean r_i and variance $\sigma^2=(0.163r_i)^2$.

Once a job i has entered the processor, the PM reduces c_a by c_i and holds the job until p_i units of time have passed. Then c_i units of processor storage are freed, that is, c_a is increased by an amount c_i. The storage used c_i and the estimated run time r_i are sent to the adaptive model as an indication of job completion.

Adaptive Model

The adaptive model, or AM, consists of an adaptive or learning algorithm which, based on data concerning completed jobs, adjusts the weights w_1, w_2 and w_3 in the RAM. The adjustments are based on comparison of actual job processing results with a predefined performance standard.

There are several possible approaches to the determination of the weight adjustments using techniques developed by researchers in the fields of pattern recognition and artificial intelligence. In the application of these techniques to the AM we consider two approaches: (1) an error-correction algorithm based on training methods used in pattern classification machines, (2) a reinforcement method used in the training of learning machines to exhibit intelligent behavior.

For each job i whose processing is completed by the PM, the AM receives the storage requirement c_i and the estimated run time r_i. The RAM provides d_i the time the job spent waiting in the input queue after generation by the JQM. Any decision made by the AM concerning changes in the RAM must be based solely on these data.

In the case of the error-correction approach, the important steps in the decision process are: (1) the determination of which jobs were not completed within the specified performance standard, (2) the selection of the adjustments to be made to correct the condition causing unsatisfactory performance.

Assume, for example, that we demand for minimum acceptable performance that $d_i \leq r_i$ for all jobs, that is, that the time d_i that job i spent waiting in the input queue must be less than or equal to the job's estimated run time r_i. Then if $d_i > r_i$ for a job i we wish to have the RAM adjusted so that for future jobs with similar c_i and r_i, the condition $d_i \leq r_i$ will be satisfied.

In order to accomplish the necessary adjustments, the error-correction algorithm must be guided by some heuristics that are based on known relationships between the weights in the RAM and performance of the system on jobs with certain general storage and estimated run time characteristics.

Four possible heuristics are: (1) if a job with short run time r_i and large storage c_i had $d_i > r_i$, increase the influence of the "short run time" allocation algorithm, (2) if a job with large run time r_i and small storage c_i had $d_i > r_i$, increase the influence of the "small storage" allocation algorithm, (3) if a job with large run time r_i and large storage c_i had $d_i > r_i$, increase the influence of the "storage times run time" allocation algorithm, (4) if a job with short run time r_i and small storage c_i had $d_i > r_i$, increase the influence of the "short run time" and "small storage" allocation algorithms.

Since w_1 is associated with the "storage times run time" allocation scheme, w_2 is associated with the "short run time" allocation algorithm and w_3 is the weight related to the "small storage" allocation method, we can state several rules for the error-correction algorithm in more symbolic form. For a job with run time r_i, storage c_i, queue wait time d_i and $d_i > r_i$, we have: (1) if $(1-r_i+c_i)/2$ is near 1, increase w_2 and decrease w_1 and w_3, (2) if $(1-r_i+c_i)/2$ is near 0, increase w_3 and decrease w_1 and w_2, (3) if $(c_i+r_i)/2$ is near 1, increase w_1 and decrease w_2 and w_3, (4) if $(c_i+r_i)/2$ is near 0, increase w_2 and w_3 and decrease w_1.

Changes in a particular weight of the RAM objective function will cause a particular allocation algorithm to have more or less influence than the other allocation schemes on which jobs are chosen from the input queue for processing.

Clearly the heuristics described must be more precisely defined in terms of exactly how the conditions that invoke a particular decision rule are satisfied and in terms of exactly how increases or decreases in the weights are to be implemented.

The principles of learning by reinforcement can also be used to improve the ability of the AM to make effective decisions concerning changes in the RAM.

This approach involves evaluation of the decisions of the AM learning machine by a trainer of some sort. If the decision resulted in an improvement with respect to a standard of performance, the use of the decision would be encouraged or positively reinforced. Similarly, if the decision resulted in a degradation of performance, use of the decision would be discouraged or negatively reinforced.

Specifically, when an input g_k to a learning machine results in a decision a_j by that machine and a_j causes an improvement in performance of the system affected by the machine, the trainer will positively reinforce decision a_j in response to input g_k. The reinforcement should be such that the probability of the learning machine making decision a_j in response to the input g_k is increased and the probability of decisions other than a_j in response to g_k is decreased. The learning machine essentially chooses decisions based on a changing conditional distribution of the decisions over the range of possible inputs.

Suppose $P_L(a_j|g_k)$ is the conditional probability that the learning machine will choose decision a_j in response to g_k as the Lth input. We can specify the set of these probabilities as shown in Figure 5 for various combinations of decision and input.

$$\begin{array}{cc} & \begin{array}{cccc} g_1 & g_2 & \cdots & g_n \end{array} \\ \begin{array}{c} a_1 \\ a_2 \\ \vdots \\ a_m \end{array} & \left[\begin{array}{c} \\ \\ P_L(a_j|g_k) \\ \\ \end{array}\right] \end{array}$$

Fig. 5--Matrix of decision|input conditional probabilities

On each occurrence of an input, the trainer computes the elements of the (L+1)th conditional probability matrix from the Lth matrix elements. If g_k was not the Lth input, then $P_{L+1}(a_j|g_k)=P_L(a_j|g_k)$ for all j. If g_k was the Lth input and decision a_j was made in response to the g_k, the trainer can either positively or negatively reinforce the decision. If the decision was correct and positive reinforcement is called for, $P_{L+1}(a_j|g_k)=\theta P_L(a_j|g_k)+(1-\theta)$ for $0<\theta<1$ and $P_{L+1}(a_i|g_k)=\theta P_L(a_i|g_k)$ for $0<\theta<1$ and $i\neq j$. If the decision was incorrect negative reinforcement can be applied by setting $P_{L+1}(a_j|g_k)=\theta P_L(a_j|g_k)$ for $0<\theta<1$ and $P_{L+1}(a_i|g_k)=\theta P_L(a_i|g_k)+(1-\theta)$ for $0<\theta<1$ and $i\neq j$.

In the context of the AM, assume that several possible decisions can be made by the learning machine: (1) increase w_2, decrease w_1 and w_3, (2) increase w_3, decrease w_1 and w_2, (3) increase w_1, decrease w_2 and w_3, (4) increase w_2 and w_3, decrease w_1, (5) do not change w_1, w_2 or w_3. Let these decisions be a_1, a_2, a_3, a_4 and a_5.

Possible inputs to the AM learning machine upon completion of job i are shown in Table 1.

input	c_i minimum	c_i maximum	r_i minimum	r_i maximum	d_i condition
g_1	50	200	0	.5	$d_i \le r_i$
g_2	50	200	0	.5	$d_i > r_i$
g_3	50	200	.5	1	$d_i \le r_i$
g_4	50	200	.5	1	$d_i > r_i$
g_5	200	350	0	.5	$d_i \le r_i$
g_6	200	350	0	.5	$d_i > r_i$
g_7	200	350	.5	1	$d_i \le r_i$
g_8	200	350	.5	1	$d_i > r_i$

Table 1--Possible AM inputs on completion of job i

Based on the heuristics discussed earlier, the trainer should positively reinforce (1) a_5 in response to g_1, g_3, g_5, g_7, (2) a_4 in response to g_2, (3) a_2 in response to g_4, (4) a_1 in response to g_6, (5) a_3 in response to g_8. The trainer should negatively reinforce all other combinations of decisions and inputs.

Consider the example of positive reinforcement of the choice of decision a_1 in response to g_6. The input g_6 represents unacceptable performance of the system ($d_i > r_i$) on a job with large storage requirement and short estimated run time. The heuristic to be applied here is to increase the influence of the "short run time" allocation algorithm in the RAM by increasing w_2; this can be accomplished by choice of decision a_1.

The actions of the trainer in the example above are based on the heuristics defined previously. The learning machine is dependent on the trainer to reinforce it; if changes occur in the distribution of the inputs the trainer must cause changes in the $P_L(a_j|g_k)$ probabilities so that the machine can adapt to the new input environment.

V. MODEL RATIONALE AND INTERPRETATION

The emphasis in the development of the system model has been on the learning machine and adaptive resource allocation aspects. Simplifications have been made in the JQM and PM to avoid some of the usual complications associated with computer job queue and processor modeling.

The job queue model defines jobs in classes according to storage and estimated run time values. This is not an unusual approach in real computer system operations, however more complex class definitions are possible and may be more efficient. For example, a third class definition parameter might be the number of input/output devices required by the job. Additional terms and constraints in the RAM and changes in the PM and AM could be implemented to accomodate this three-parameter job class scheme. Essentially this definition would add complexity to the system model but is not likely to require changes in the underlying learning or adaptive principles.

The assumption in the JQM of a uniform distribution of jobs within a job class allows specification of a fairly small set of heuristics to aid the learning machine decision-making process. This is probably an unrealistic assumption as compared with actual job storage and estimated run time distributions for real computer system operations.

The processor could include such complications as input/output device contention, priority schemes, more realistic run time variations, queuing effects, storage partitioning or virtual storage. In our development of the system model we have regarded the PM as a "black box" that introduces a delay or acceleration into the passage of a job from the RAM to the AM. In other words, a job may be ended by the PM and sent to the AM before another job that was begun earlier is completed due to the variation in run time. This reordering of the jobs received in a particular sequence by the processor is common in any multiprogramming computer system.

The parameter $c_{a_{max}}$, the size of the system processor, has a significant effect on the RAM constraint. Study of the result of variations in this size would be important in any evaluation of the system model.

The objective function of the integer program in the resource allocation model can be interpreted as a learning machine discriminant function.[1] A 2n-dimensional space is defined by the c_i and r_i parameters for each of the n jobs in the input queue. If in the expression

$$F = w_1 \sum_{i=1}^{n} (c_i/350) r_i x_i + w_2 \sum_{i=1}^{n} (-r_i x_i) + w_3 \sum_{i=1}^{n} (-c_i/350) x_i,$$

we regard the x_i and w_1, w_2, w_3 as parameters, the expression

---

[1]Nils J. Nilsson, Learning Machines (New York, New York: McGraw-Hill Book Company, 1965), pp. 6-8.

can be described as a linear combination of linear and second-order terms in the c_i and r_i. If n is 2 we can write

$$F = (w_1x_1/350)c_1r_1 + (w_1x_2/350)c_2r_2 + (-w_2x_1)r_1 + (-w_2x_2)r_2 + (-w_3x_1/350)c_1 + (-w_3x_2/350)c_2.$$

Thus, F contains some hyperbolic and some linear terms which, depending on the weights and the parameters x_i, can be regarded to partition the 2n-dimensional space of c_i and r_i variables into two subspaces separated by a combination of hyperboloid and hyperplane surfaces.

The separating hypersurface defined by the objective function is altered during the solution of the integer program by testing of alternative feasible x_i solution sets, which causes certain terms of the linear combination to be included or deleted depending on whether x_i is 0 or 1 for the terms. Note that the constraint of the program is a hyperplane in the 2n-dimensional c_i, r_i space.

One of the two subspaces separated by the objective function hypersurface contains jobs to be executed when the optimum feasible x_i have been determined.

There are many alternatives to the error-correction and reinforcement learning approaches suggested for the adaptive model. Heuristic programming and problem-solving methods in artificial intelligence seem to be receiving a large amount of support and attention.[1] However, since any model

[1]Edward A. Feigenbaum, Artificial Intelligence: Themes in the Second Decade, Stanford University Report AI-67 (Stanford, Calif: Stanford University, 1968), pp. 5-18.

of a computer system that attempts to reflect "real world" conditions will probably contain at least one statistical element, these popular techniques may not be applicable in the use of artificial intelligence to improve computer system performance.

In the heuristic programming and problem-solving approaches a representation is required that defines the problem space over which the search for a solution is conducted. This representation problem is sometimes a difficult one and poor problem representation can lead to extremely inefficient searching.[1]

When a statistically defined process (which may be based on empirical data) is involved as in computer system modeling, it is not clear that an adequate representation can be defined that will allow application of search methods. Uncertainty as to how to specify the parameters of the statistical processes may make any representation such as a search graph difficult to construct and verify.

Both approaches suggested for the AM learning algorithm essentially involve feedback to the resource allocation model. Due to the statistical processes represented by the JQM and PM, the AM may be receiving inputs that have significant fluctuations. It might be useful to include some type of preprocessor in the AM so that instead of altering the RAM

---

[1]Edward A. Feigenbaum, Ibid., pp. 27-31.

based on system performance for each job, it would respond to some "average" performance for a group of jobs.

LIST OF REFERENCES

1. Willis H. Ware, Limits in Computing Power. Rand Corp. Paper P-4710. Santa Monica, Calif: Rand Corp., 1971.

2. Edward A. Feigenbaum and Julian Feldman,eds., Computers and Thought. New York: McGraw-Hill Book Company, 1963.

3. Richard A. Arnold, et al., Mathematical Models of Information Systems. USAF Rome Air Development Center Report RADC-TR-66-37. Griffiss Air Force Base, New York: Rome Air Development Center, 1966.

4. George S. Fishman and Philip J. Kiviat, Digital Computer Simulation: Statistical Considerations. Rand Corp. Memorandum RM-5387-PR. Santa Monica, Calif: Rand Corp., 1967.

5. Edward A. Feigenbaum, Artificial Intelligence: Themes in the Second Decade. Stanford Univ. Report AI-67. Stanford, Calif: Stanford Univ., 1968.

6. Peter E. Hart and Richard O. Duda, Survey of Artificial Intelligence. Menlo Park, Calif: Stanford Research Institute, 1971.

7. Aerospace Technology Division, Library of Congress, Learning, Self-Learning, and Pattern Recognition. Library of Congress Report ATD 67-64. Washington, D.C.: Library of Congress, 1969.

8. Nils J. Nilsson,Learning Machines. New York: McGraw-Hill Book Company, 1965.

9. Nils J. Nilsson,Problem-Solving Methods in Artificial Intelligence. New York: McGraw-Hill Book Company, 1971.

10.Paul L. Meyer,Introductory Probability and Statistical Applications. 2d ed. Reading, Massachusetts: Addison-Wesley Publishing Company, 1970.