LA-7475-MS

Informal Report

The Application of Artificial Intelligence Techniques to the Acceleration of Monte Carlo Transport Calculations

LOS ALAMOS SCIENTIFIC LABORATORY
Post Office Box 1663 Los Alamos. New Mexico 87545

UNITED STATES
DEPARTMENT OF ENERGY
CONTRACT W-7405-ENG. 36

LA-7475-MS
Informal Report
UC-32
Issued: September 1978

The Application of Artificial Intelligence Techniques to the Acceleration of Monte Carlo Transport Calculations

J. L. Macdonald
E. D. Cashwell

CONTENTS

THE APPLICATION OF ARTIFICIAL INTELLIGENCE TECHNIQUES TO THE ACCELERATION OF MONTE CARLO TRANSPORT CALCULATIONS

by

J. L. Macdonald
E. D. Cashwell

ABSTRACT

The techniques of learning theory and pattern recognition are used to learn splitting surface locations for the Monte Carlo neutron transport code MCN. A study is performed to determine default values for several pattern recognition and learning parameters. The modified MCN code is used to reduce computer cost for several non-trivial example problems.

I. INTRODUCTION

The objective of this research is to apply the artificial intelligence techniques of learning theory[1] and pattern recognition[1,2] to a general-purpose Monte Carlo particle transport code. Together the techniques are used to reduce computer cost by learning suitable locations for splitting surfaces, a function usually considered to require human intuition and experience.

The purpose of this introductory section is to describe the problems encountered with Monte Carlo computer cost reduction, explain what is meant by an "artificial intelligence" approach, and summarize the remainder of this report.

A. Computer Cost Reduction of Monte Carlo Calculations

The Monte Carlo method has been used since the early 1940s[3] for the solution of particle transport problems. In the past it has gained a reputation of being computationally expensive and "a method of last resort."[4] However, with the increasing speed of computers and the increasing demand for more exact three-dimensional transport calculations, the method is becoming more attractive and has become a commonplace tool for the solution of particle transport problems.

Regardless of the speed of a computer calculation, there will always be a demand for acceleration. This is due not only to the desire for reducing computer costs, but also to the fact that problems previously considered unsolvable can be solved. There is certainly no shortage of techniques for reducing the computer time of Monte Carlo calculations (see Bibliography of Ref. 5). However, when one looks at the techniques routinely used, the number is small. This is primarily for the following reasons:

(1) Many techniques are developed and demonstrated for very simple problems. When one tries to apply such a technique to "real world" problems, complications arise which cause the technique to lose its effectiveness.

(2) Some techniques are designed primarily to reduce the variance of a calculation. Although they may be quite successful in this respect, they may reduce computer costs only slightly.

(3) A technique may not be used because it is too difficult to use or too easy to misuse. Techniques with this drawback usually require quantitative input by the user for which he has no intuition. If the right values are used, appreciable savings can result. However, if the wrong values are selected, not only can computer cost reduction be negligible, but the technique may bias the results.

(4) Another drawback of a technique may be its range of application. Although the technique may not have any of the previous drawbacks, it may apply to so few problems that its overall impact is small.

One of the easiest ways to reduce computer costs is to select an efficient estimator. Point detectors[5] and track length tallies[5] can save large amounts of computer time with little user input.

Once a suitable estimator has been chosen, the computer cost reduction problem usually consists of directing particles into that region of state space (phase space) which must be sampled adequately before a reasonable error can be obtained. One of the oldest and most popular techniques for doing this is geometry splitting accompanied by Russian roulette (see Sec. II.A). Although quite effective, geometry splitting does have the disadvantage of being limited to the geometric regions of the problem. In addition, it requires that the user provide quantitative values for the importance of cells.

B. The Artificial Intelligence Approach

The first phase of this research consists of generalizing geometry splitting to include all state space variables in any combination or functional form

desired by the user. The practical problem of where to locate the splitting surfaces is the main topic of this research.

One solution to this problem is to solve the adjoint transport problem for the importances. Although satisfactory for some problems, this approach is not only more complicated than a single forward calculation but it can, for some problems, result in a net increase in computer costs.

If the user could use all the information generated during a calculation, he would greatly increase his chances of selecting suitable splitting surfaces. He would recognize from which regions of state space tally contributions were coming from, allowing him to make quantitative decisions on where to locate splitting surfaces. However, since Monte Carlo calculations are not generally run interactively, the user is not aware of this information.

The artificial intelligence approach consists of replacing the human user by a system which learns the surface locations by using the same information the user would. Such a system consists of a pattern recognition scheme with a controller for monitoring the learning. With such a heuristic approach to a problem, there is no mathematical analysis and thus no proof of computer costs reduction. This should not alarm the user since such proofs are rarely if ever found for the general Monte Carlo problem even when using the most rigorously derived techniques.

C. Guide to this Report

Section II of this report describes the theory of splitting, pattern recognition, and evaluation of computer costs. For more detailed information on these topics the reader is referred to Ref. 6.

Before a technique can be realistically evaluated, it must be implemented in a multipurpose code. Only then do the practical problems of its application become apparent. The neutron transport code MCN[7] was chosen for this research. Section III describes the modifications necessary to use state space splitting and pattern recognition. This section is oriented towards the reader who is interested in the actual mechanics of the techniques and can be skipped by the casual reader.

In the development of the pattern recognition and learning control systems, many parameters arise whose values are undetermined for the general problem. In Sec. IV, a sensitivity study is performed for these parameters and default values are selected.

The use of the technique is demonstrated in Sec. V using a non-trivial Monte Carlo problem and the parameter values found in Sec. IV.

Finally, Sec. VI summarizes the results and presents some recommendations for further use and improvements.

II. THEORY

The goal of this research is to use pattern recognition techniques to learn splitting surfaces in a Monte Carlo calculation and to determine the computer savings of such a system. The purpose of this section is to explain the three phenomena mentioned above, i.e., (1) splitting in a Monte Carlo calculation, (2) pattern recognition, and (3) computer savings. These subjects will be described to the extent necessary to understand this research. In the description of splitting, it is assumed that the reader has some familiarity with the Monte Carlo process as used for particle transport. If this is not the case, the reader is referred to Ref. 5. No knowledge of pattern recognition is assumed; however, if more information is desired, there are several texts available on pattern recognition.[1,2,8,9,10] The notation used in this section is summarized in Appendix B.

A. Splitting in a Monte Carlo Calculation

This section describes the processes of splitting and Russian roulette, both as they are now used in LASL Monte Carlo codes[11] and as they are used for state space splitting. No attempt will be made in this report to describe why splitting works. This information can be found in Ref. 5. Instead, emphasis will be placed on describing how these functions are performed.

1. Geometry and Energy Splitting. Geometry splitting, accompanied by Russian roulette, is one of the most commonly used variance reduction techniques. It consists of dividing the geometry of the problem into regions and assigning an importance to each region. This "importance" is selected so that particles in a region of high importance have a higher probability of contributing to a Monte Carlo tally. A particle of weight wt going from a region of low importance I_n to one of greater importance I_{n+1} is split at the boundary between the regions into I_{n+1}/I_n particles, each of weight $wt(I_n/I_{n+1})$. (The weight of a particle represents a fraction of a particle, which may be greater or less than 1.) A particle leaving a region of importance I_n and entering a region of lesser importance I_{n+1} survives with probability I_{n+1}/I_n and weight $wt(I_n/I_{n+1})$.

Figure II.1 shows an example of splitting planes and importance regions used with a semi-infinite slab of thickness T. For problems in which T is many mean free paths, splitting a Russian roulette can be very effective and can lead to several orders of magnitude reduction in computer time.

Splitting and Russian roulette can also be used in "energy space" for problems in which particular energy regions are more important than others. An example of energy splitting is the tallying (or editing) of U^{235} thermal fission. In this case, one would separate energy space into regions which increase in importance as thermal energies are approached as shown in Fig. II.2. Particles are split when they lose energy and cross an energy splitting surface. In the LASL Monte Carlo codes, particles do not undergo Russian roulette because of an increase in energy (due to upscattering, fission, etc.).

The popularity of the above techniques can be attributed primarily to the ease of their implementation. In most cases a guess based on intuition will lead to a large savings in computer time. Usually the importance regions specified are already geometrically defined by the problem (different materials, densities, and shapes) and the user only has to provide the importances.

One problem with these techniques is that they require an educated guess on the part of the user as to where splitting surfaces should be located. Although rough guesses help, several "trial and error" Monte Carlo runs are required if any optimization is to be attempted. Another problem is that these techniques allow the user to discriminate in energy and spatial coordinates only. Time and directional variables are not used. Furthermore the energy and spatial variables are considered only in an independent manner.

2. State Space Splitting. Consider the general Monte Carlo problem in which particles are characterized by the following state variables:

(a) Spatial coordinates - x, y, and z
(b) Angular coordinates - u, v, and w, where these values are the cosines of the particle line-of-flight with the x, y, and z axes, respectively
(c) Energy - E
(d) Time - t.

In state space splitting, all variables can be used to determine which regions in state space are more important than others. A practical problem arises in determining the importances of these state space regions. Users have enough difficulty with the three spatial coordinates and energy; the complexity

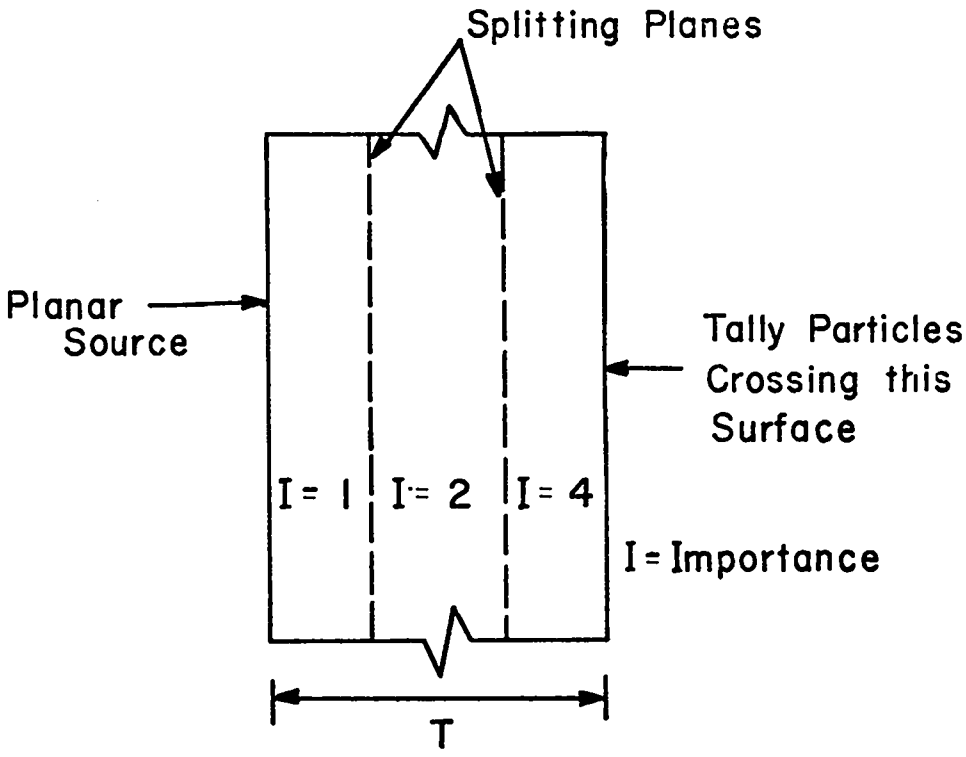

Fig. II.1. Geometry Splitting

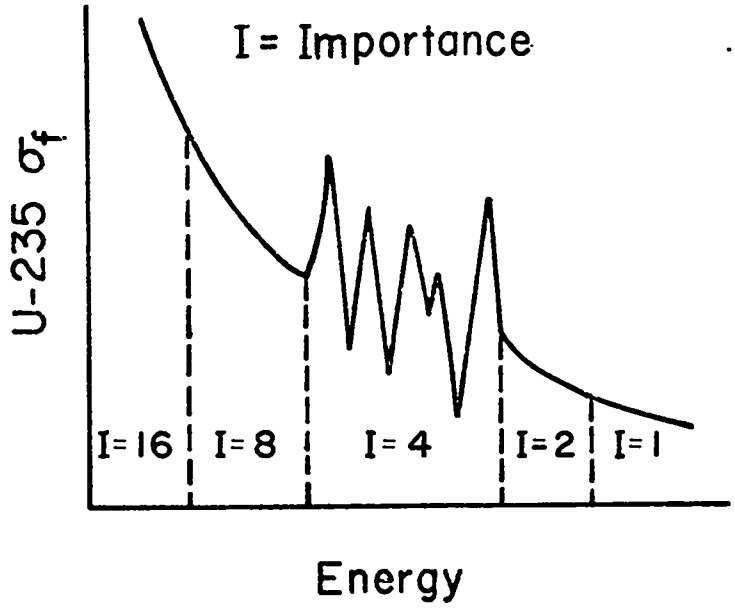

Fig. II.2. Energy Splitting

involved in determining splitting surfaces in eight (seven independent) dimensions would certainly confuse even the most experienced user.

A point in state space can be identified by the eight-component vector

$$\underline{X} = \begin{bmatrix} x \\ y \\ z \\ u \\ v \\ w \\ E \\ t \end{bmatrix} \tag{II.1}$$

Seven components would be sufficient since only two of the three directional variables are independent (i.e., $u^2 + v^2 + w^2 = 1$). In order to split, one must know to which importance region $\underline{X}$ belongs. This problem can be simplified somewhat if state space is first transformed to a more efficient space. The resulting space is termed feature space[2] where each component of feature space consists of a function of state space variables. A point in feature space is represented by the feature vector

$$\underline{F} = \begin{bmatrix} f_1(x,y,z,u,v,w,E,t) \\ f_2(x,y,z,u,v,w,E,t) \\ \vdots \\ f_R(x,y,z,u,v,w,E,t) \end{bmatrix} \tag{II.2}$$

The transformation of state space into feature space is beneficial for several reasons:

(a) State space variables which are not important in a problem can be eliminated.

(b) Functions of state space are often easier to relate to an importance region. For example, the distance from a neutron located at (x,y,z) to a tally located at (x*,y*,z*) is given by

$$f_i = d = \sqrt{(x - x^*)^2 + (y - y^*)^2 + (z - z^*)^2} \ .$$

(c) As a result of (a) and (b), the dimensionality of feature space R can be made less than seven.

(d) Feature space allows the normalization of state space. Thus variables with different units (time in seconds, energy in MeV, distance in cm) and different ranges can be normalized.

(e) A surface which may have to be represented by a quadratic in state space can often be linearized in feature space.

The problem then is the following: given a vector $\underline{X}$ and transforming it to vector $\underline{F}$, find to which importance region $\underline{F}$ belongs. This problem is solved by the introduction of "discriminant functions."[2] A discriminant function g is defined so that two regions in feature space are separated by the surface

$$g(f_1, f_2, \ \ldots \ f_R) = 0. \qquad \text{(II.3)}$$

In Monte Carlo problems, feature space is arranged so that for N_c regions of importance, there will be $N_c - 1$ discriminant functions separating the regions. In this research, only linear discriminant functions will be considered, so that

$$g(f_1, f_2, \ \ldots, \ f_R) = w_1 f_1 + w_2 f_2 + \ \ldots \ + w_{R+1} = \underline{W} \cdot \underline{F^*} \ , \qquad \text{(II.4)}$$

where

$$\underline{F^*} = \begin{bmatrix} f_1 \\ f_2 \\ \cdot \\ \cdot \\ \cdot \\ f_R \\ 1 \end{bmatrix} \text{ and } \underline{W} = [w_1 w_2 \ \cdots \ w_{R+1}]. \qquad \text{(II.5)}$$

The vector $\underline{W}$ is called the weight vector[2] and can be learned by a pattern recognition system or supplied by the user. The vector $\underline{F^*}$ is referred to as the augumented feature vector.

If the importance regions or "classes" of feature space are ordered such that 1 is the least important and N_c is the most important and, for a fixed i on the range $1 \leq i \leq N_c - 1$, if

$$g_i(\underline{F}^*) > 0,$$

where g_i is the discriminant function between regions i and i + 1, $\underline{F}^*$ must belong to some class j, where $N_c \geq j \geq i + 1$. Similarly, if

$$g_i(\underline{F}^*) < 0,$$

$\underline{F}^*$ must belong to a class j, where $1 \leq j \leq i$. By substituting $\underline{F}^*$ into the g_i, the class of $\underline{F}^*$ can be identified.

In this research, the importances of classes 1,2, ... N_c are given the values $1,2,2^2, \ldots, 2^{N_c-1}$. When a particle undergoes a collision or intersects a surface in a Monte Carlo problem it goes from one point in state space to another. Consequently, it goes from one $\underline{F}_n$ in feature space to another, $\underline{F}_{n+1}$. Using the above procedure for identifying the classes to which $\underline{F}_n$ and $\underline{F}_{n+1}$ belong, the importance ratio

$$A_{n,n+1} = \frac{I_{n+1}}{I_n}, \qquad \text{(II.6)}$$

where I_n is the importance of the region to which $\underline{F}_n$ belongs, can be calculated. This ratio is used as follows:

if $A_{n,n+1} = 1$, particle is transported unchanged
if $A_{n,n+1} < 1$, particle undergoes Russian roulette and is killed with probability $(1-A_{n,n+1})$
if $A_{n,n+1} > 1$, particle is split into $A_{n,n+1}$ particles.

In this research, the selection of a functional form for the features is a human task with the following constraints:

(a) All features must be normalized between 0 and 1

(b) Features must be oriented such that as the feature increases from 0 to 1,

the importance of feature space increases. This results in the most important point of feature space being located at (1 ... 1) and the least important point at (0 ... 0).

The weight vectors $\underline{W}_i$, $i = 1, \ldots, N_c - 1$, for the $N_c - 1$ discriminant functions can be supplied by pattern recognition or by the user.

In summary, as a particle in a Monte Carlo calculation travels through state space it enters regions of different importance. The ratios of these importances will decide whether the particle should undergo splitting, Russian roulette, or continue unchanged.

B. Pattern Recognition

The purpose of this section is to describe a pattern recognition system suitable for learning state space splitting surfaces.

Before anything can be learned, a pattern recognition system must be provided with information from a "teacher." Such information consists of a series of "prototypes."[2]

As was mentioned in the previous section, it is the weight vector $\underline{W}$ that is learned. This learning will take place by using prototypes and a suitable pattern classification algorithm.

In addition to prototypes and classification algorithms, the learning system must be able to coordinate the learning of several surfaces at once, determine when a surface has been learned adequately, and when a surface should be used for splitting. These functions are performed by a control system.

1. Prototypes. A prototype consists of a state space vector $\underline{X}$ (or the corresponding feature vector $\underline{F}$) and the known classification of $\underline{X}$ (or $\underline{F}$). These prototypes are used in the learning process as shown in Fig. II.3. The vector $\underline{W}_{i,j}$ is the weight vector of the i'th discriminant function after the j'th iteration (each prototype produces an iteration).

Prototypes are extracted from the calculation in the following manner:

(1) At the j'th collision or surface intersection a feature vector, $\underline{F}_j$, is generated and stored. The weight of the particle wt_j, and its tally weight contribution t_j, are also stored. The t_j is given by

$$t_j = \sum_{b=1}^{B} D_b \cdot wt_j \cdot \delta_{b,j} , \qquad \text{(II.7)}$$

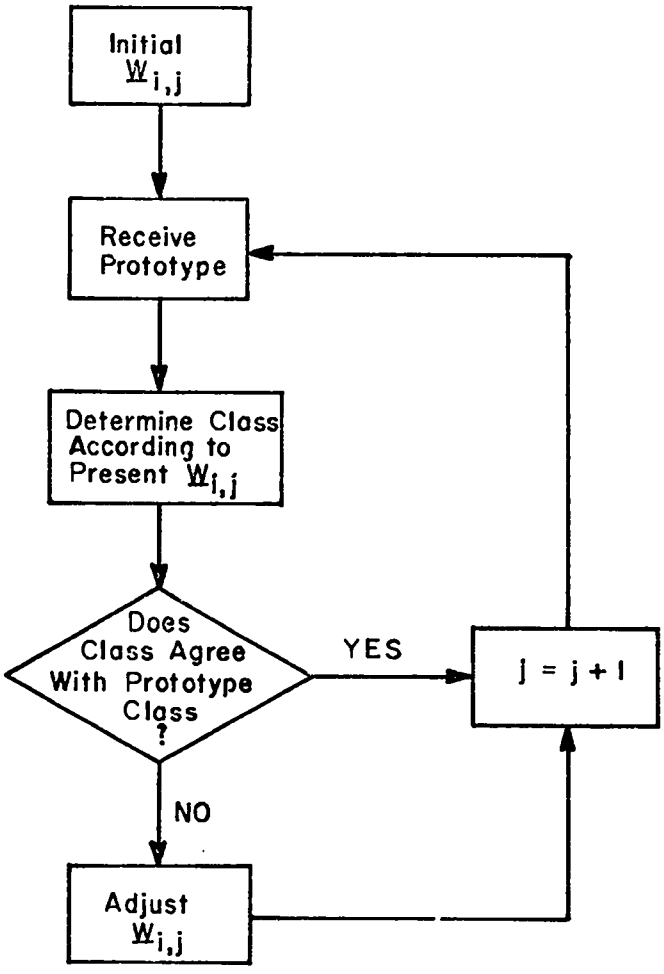

Fig. II.3. Learning the Weight Vector

where B = number of tally bins[†]

$\delta_{b,j}$ = 1, if a contribution was made to the b'th tally bin before the j'th collision (or surface intersection) but after the (j-1)'th

= 0, if not

D_b = a weighting factor applied to the b'th tally bin. D_b is used to direct particles preferentially to specific tally bins.

[†]A tally bin is the lowest level of the tally data structure. For example, for a surface current tally, a particle crossing a surface at an angle $\cos^{-1}\mu$, with energy E, at time t is placed in bin ℓ,i,j where

$$\mu_{\ell-1} < \mu \leq \mu_{\ell}$$
$$E_{i-1} < E \leq E_i$$
$$t_{j-1} < t \leq t_j$$

and μ_{ℓ}, E_i, and t_j are tally bin limits specified by the user.

(2) When the present particle "track"[†] is terminated, the total tally weight contribution T_j is evaluated

$$T_j = \frac{\sum_{k=j}^{N_p} t_k}{wt_j} \quad , \tag{II.8}$$

where N_p = the number of prototypes created since the birth of the present source particle.

(3) There are two types of splitting surfaces used in this research. The first type separates prototypes (or classes containing prototypes) for which $T_j = 0$. The second type separates prototypes for which $T_j \neq 0$. Let the number of $T_j \neq 0$ surfaces be given by N_u and the number of importance regions (or classes) by N_c. Then surfaces $(N_c - N_u)$ through $(N_c - 1)$ are the N_u surfaces used to separate $T_j \neq 0$ prototypes and surfaces 1 through $(N_c - N_u - 2)$ separate $T_j = 0$ prototypes. Surface $(N_c - N_u - 1)$ separates $T_j = 0$ prototypes from $T_j \neq 0$ prototypes.
If $N_u \neq 0$ and $T_j \neq 0$, T_j is compared to a predetermined set of class weight contribution thresholds,

$$\hat{T}_i \quad i = (N_c - N_u), \ldots, (N_c - 1).$$

If $T_j \leq \hat{T}_i$, then the j'th prototype belongs to some class C, where $C \leq i$. If $0 < T_j \leq \hat{T}_L$ $(L = N_c - N_u)$, $C = N_c - N_u$. If $T_j = 0$, C must be determined as if $N_u = 0$ (see below) since it is only known that $C < (N_c - N_u)$.
If $N_u = 0$ and $T_j \neq 0$, C is set to N_c. If $T_j = 0$, the highest importance class, C_H, obtained by any prototype $K (K = j + 1, \ldots, N_p)$ is determined by the discriminant functions g_i (see Sec. II.A.2). The class C is then set to $C_H - 1$. If $C_H = 1$, C is set to 1.

[†]A "track" is created at the source, from fission, from (n,xn) reactions, or from splitting.

This procedure for determining the classification of a prototype is more easily described in terms of the actual programming logic used. This is done in Sec. III.C.1.

(4) The vector $\underline{F}$ and the class C together provide an information point or "prototype" to be used by the pattern classification algorithm.

2. Pattern Classification Algorithm. Numerous pattern classification algorithms, both statistical and deterministic, were investigated in this research. The unsatisfactory techniques are not discussed in this report. Only the algorithm which was found suitable for learning Monte Carlo splitting surfaces is described.

Consider a single linear splitting surface in feature space given by

$$w_1 f_1 + w_2 f_2 + \ldots w_{R+1} = 0 ,$$
$$\underline{W} \cdot \underline{F}^* = 0$$

where R is the dimensionality of feature space. The vector $\underline{W}$ is adjusted after the j'th prototype as given by

$$\underline{W}_{j+1} = \underline{W}_j + c\, \underline{F}^*, \qquad \text{(II.9)}$$

where

$$c = 0 \text{ if } NT = NS$$
$$c < 0 \text{ if } NT < NS$$
$$c > 0 \text{ if } NT > NS.$$

NT = the importance class according to the prototype or "teacher" class

NS = the importance class according to the present discriminant functions or "student" class.

This algorithm is the general form of the sequential deterministic classifier[2] which is guaranteed to converge if the prototypes are non-overlapping and linearly separable.[2] These requirements are discussed below.

A feature vector $\underline{F}$ has associated with it a probability $P(\underline{F}|n)$ of belonging to class n (according to the teacher), where $n = 1, \ldots, N_c$. Consider the example of a feature vector $\underline{F}^1$ that occurs 100 times in prototypes. If NT = 2 in 80 prototypes, and NT = 1 in 20 prototypes, then

$$P(\underline{F}^1|1) = .2$$
$$P(\underline{F}^1|2) = .8.$$

For any vector $\underline{F}$,

$$\sum_{n=1}^{N_c} P(\underline{F}|n) = 1.$$

In order for classes to be non-overlapping, there must exist an n such that

$$P(\underline{F}|n) = 1 \quad 1 \leq n \leq N_c$$

and

$$P(\underline{F}|i) = 0 \quad i = 1, \ldots, N_c \; i \neq n,$$

for all $\underline{F}$. The classes for Monte Carlo calculations will never satisfy this condition since each classification of a prototype NT is only a single estimate and can vary statistically.

Consider a two-class problem ($N_c = 2$). If a linear surface exists such that for all $\underline{F}$ on one side of the surface,

$$P(\underline{F}|1) = 1$$

and for all $\underline{F}$ on the other side,

$$P(\underline{F}|2) = 1,$$

the classes are said to be linearly separable. If classes are overlapping, they cannot be linearly separable. Thus both requirements for convergence are not satisfied by the classes used in a Monte Carlo calculation.

Although the above algorithm is not guaranteed to converge, it can still be used statistically to minimize prototype misclassifications. Instead of using $\underline{W}_{j+1}$ for splitting, the average value of $\underline{W}_{j+1}$ after J prototypes is used as given by

$$\underline{\overline{W}}_J = \frac{\sum_{j=1}^{J} \underline{W}_j}{J} = \frac{\underline{S}_J}{J} \quad . \tag{II.10}$$

The sum $\underline{S}_J$ is given by

$$\underline{S}_J = \sum_{j=1}^{J} \underline{W}_j = J \cdot \underline{W}_o + \sum_{j=1}^{J} \sum_{i=1}^{j} \Delta \underline{W}_i$$

$$= J \cdot \underline{W}_o + \sum_{j=1}^{J} \underline{H}_j \tag{II.11}$$

$$\underline{S}_J = J \cdot \underline{W}_o + \underline{R}_J \quad ,$$

where

$\underline{W}_o$ is the weight vector before the first prototype and

$\Delta \underline{W}_i$ is the weight vector change due to the i'th prototype.

From Eq. (II.9), $\Delta \underline{W}_i = c \, \underline{F}_i^*$, where $\underline{F}_i^*$ is the i'th augmented feature vector. The average splitting surface location can be calculated after J prototypes provided $\underline{H}_J$ and $\underline{R}_J$ are stored after each misclassified prototype. (For correctly classified prototypes, $\Delta \underline{W}_i = 0$.)

The magnitude of c in Eq. (II.9) determines the rate of learning. The functional form determines what quantity is to be minimized. In this research, c is given by

$$c = \lambda f_a f_b \quad , \tag{II.12}$$

where

λ controls the rate of learning,

$f_a = \frac{1}{|\underline{F}^*|}$ normalizes the adjustment, and

$f_b = \frac{1}{P(\underline{F})}$ corrects for the uneven distribution of prototypes.

Without normalization of the adjustment, prototypes with large $|\underline{F}^*|$ have more influence than those with smaller $|\underline{F}^*|$. In this research, it is desired to treat prototypes equally, regardless of their $|\underline{F}^*|$.

The function $P(\underline{F})$ is the probability distribution of prototypes in feature space. If prototypes are evenly distributed, $P(\underline{F}) = 1$. In Monte Carlo calculations, prototypes are created at some regions of feature space more than others. For example, a point source creates prototypes at a single geometric location. If f_b were not used, frequently sampled regions of feature space would dominate the learning of the splitting surface. In the general problem, $P(\underline{F})$ is not known and must be learned during the calculation. This is done by dividing each feature component axis into ten increments. The resulting feature space is divided into 10^R frequency regions (R is the dimensionality of feature space). When a prototype is being processed, the frequency region is determined and the number of prototypes in the region is incremented. The ratio of this number to the total number of prototypes is used as an approximation to $P(\underline{F})$.

C. Computer Savings

The variance of a Monte Carlo estimate[5] is given by

$$\sigma^2 = \frac{\sigma_s^2}{N} , \qquad \text{(II.13)}$$

where N is the sample size and σ_s^2 is the variance of the Monte Carlo sample.[5] The σ^2 can be reduced by either increasing the sample size or decreasing σ_s^2.

Another factor of interest in a Monte Carlo calculation is the computer time spent in processing a single particle t_p. The value of t_p is found from

$$t_p = \frac{T_c}{N} , \qquad \text{(II.14)}$$

where T_c is the total time spent on the calculation and N is again the number of particles or sample size. The number of particles required to achieve a desired σ^2 is given by Eq. (II.13) and is

$$N = \frac{\sigma_s^2}{\sigma^2} .$$

The time required to perform the calculation to obtain the variance σ^2 is given by

$$T_c = Nt_p = \frac{\sigma_s^2}{\sigma^2} t_p = \frac{1}{\sigma^2} \left(\sigma_s^2 t_p\right). \qquad \text{(II.15)}$$

The goal of this research is to reduce computer cost for a given σ^2, the cost being proportional to computer time. Thus the measurement used in this research for the cost of a calculation is given by $\left(\sigma_s^2 t_p\right)$. This quantity can be used for comparing calculations resulting in a relative cost as given by

$$R_{cost} = \frac{\left(\sigma_s^2 t_p\right)_w}{\left(\sigma_s^2 t_p\right)_{wo}}, \qquad \text{(II.16)}$$

where the subscripts refer to values obtained when running the same calculation with and without variance reduction.

III. IMPLEMENTATION

The Monte Carlo neutron transport code MCN[7] is used in this research. The purpose of this section is to describe the modifications to MCN required to perform state space splitting and pattern recognition. Although no two Monte Carlo codes are identical, these modifications should provide the reader with the necessary information to modify other codes.

Section III.A gives a brief description of the MCN code logic prior to any changes. Section III.B describes the logic that is necessary for state space splitting alone. Section III.C describes the logic for the addition of the pattern recognition system. Finally, Sec. III.D describes the logic required for controlling the learning system. References are frequently made to FORTRAN variables. These variables are summarized in Appendix C.

A. Description of MCN

The logical flow of MCN that is relevant to this research is illustrated in Fig. III.1. Each operation is identified by a number and is briefly described below.

1. Calculate Source Parameters. This operation produces a neutron at some point (x,y,z), with a direction (u,v,w), an energy E, at time t, and with weight wt.

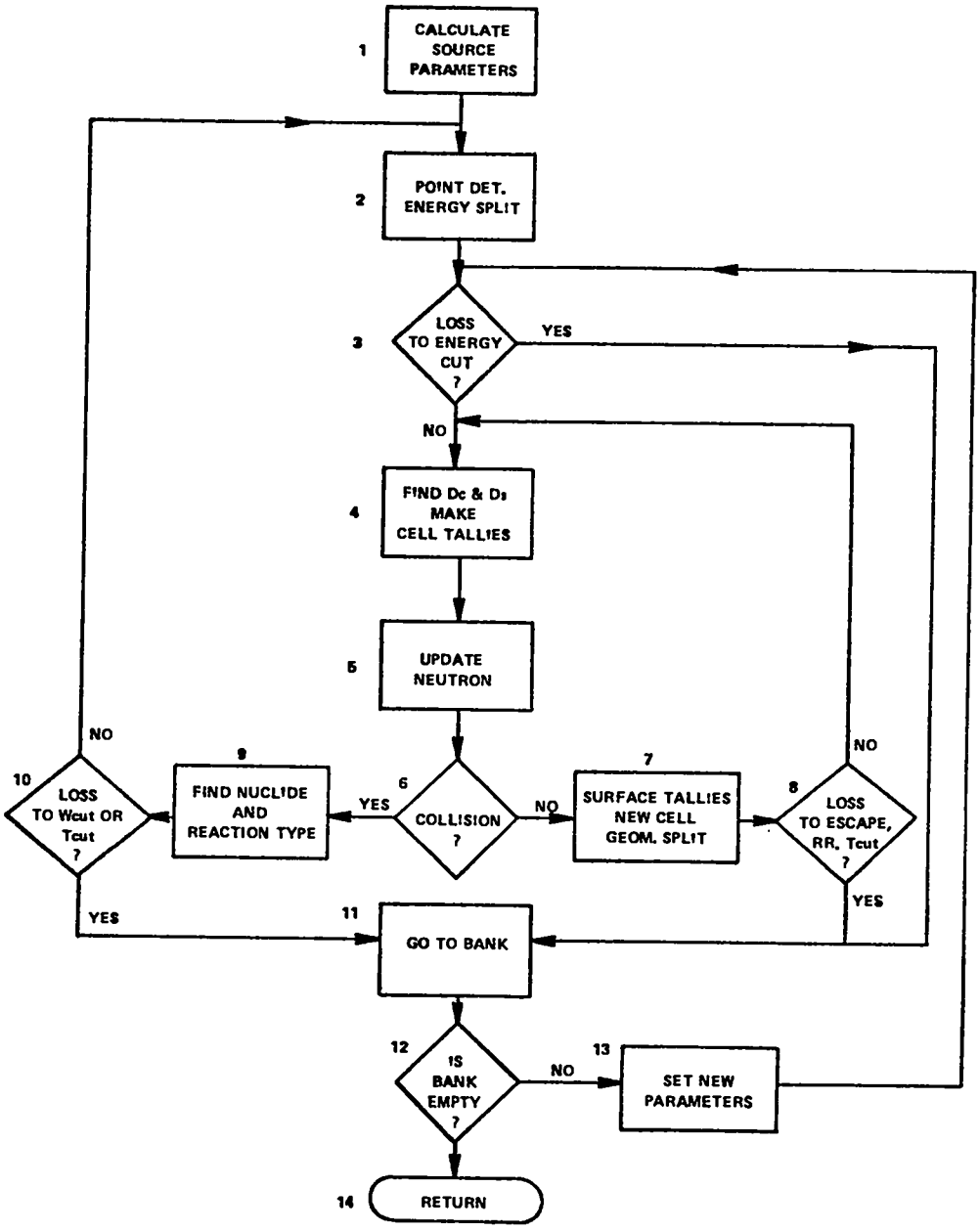

Fig. III.1. The MCN Code

2. Point Detector and Energy Split. The point detector is a tally (or edit) in MCN used for calculating flux at a point.[5] After each collision and at the source, the flux contribution of a neutron scattering at a point in space (x_o, y_o, z_o) toward the detector and reaching it without further collision is calculated. The neutron then continues with its normal transport. Energy splitting was explained earlier. When a neutron undergoes energy splitting, it is split into N_e neutrons (N_e is determined by the input), each with a weight wt/N_e (wt

is the neutron's weight before splitting). One of the N_e neutrons continues with the transport; $(N_e - 1)$ neutrons are sent to the bank (see 11 below).

3. Energy Cut. If a neutron's energy E falls below a predetermined level E_{cut} the neutron's transport is terminated.

4. Collision and Surface Distances, Cell Tallies. The distance D_s from a neutron's present position (x,y,z) along its flight path (u,v,w) to the nearest geometric surface intersection is calculated. The distance to the next collision D_c is then sampled. The shorter of these distances is used in any track length cell tallies selected by the user.

5. Update Neutron. At this point, the neutron is moved to its new location (x,y,z) as determined by D_c or D_s. The time t of the neutron is also incremented.

6. Collision. If $D_c < D_s$, a collision occurs. If $D_c > D_s$, the neutron is transported to a geometric surface.

7. Surface Tally, New Cell, Geometry Split. When a neutron crosses a surface, any surface tallies specified by the user are updated. The code then determines which geometric region or cell the neutron is entering. If the new cell has a greater importance than the old, the neutron is split into N_g neutrons, each with weight wt/N_g. One neutron is allowed to continue, $(N_g - 1)$ are banked. If the importances are the same, the neutron continues unchanged. If the new importance is less, Russian roulette is played to see if the neutron is killed or allowed to continue with increased weight.

8. Loss to Escape, Russian Roulette, or Time Cutoff. If the new cell has zero importance, if the neutron is lost to Russian roulette, or if the time t of the neutron is greater than a predetermined T_{cut}, the neutron's transport is terminated. If not, the neutron's transport continues.

9. Find Nuclide and Reaction Type. The nuclide with which the neutron collided is first determined. The weight wt of the neutron is reduced by the capture probability. The reaction type (fission, inelastic, or elastic scattering) is then determined. If additional neutrons are born (fission and (n,xn)), the extra neutrons are banked. A new energy (E) and direction vector (u,v,w) is calculated.

10. Loss to Weight or Time Cutoffs. If the time is greater than T_{cut}, the neutron's transport is terminated. If the neutron's weight is below a predetermined cutoff W_{cut}, it undergoes Russian roulette. If it is lost to Russian roulette, the history is terminated.

11. The Bank. Whenever a neutron is split (see operations 2 and 7) or undergoes multiplication from fission and (n,xn) (see operation 9), separate "tracks" are created. These tracks are placed in a bank until the present neutron has completed its transport. At this time the bank is checked to see if any tracks remain.

12. Bank Empty. If the bank is empty, the transport process for this source neutron is finished and control is returned to the non-transport functions of the code.

13. Set New Parameters. If the bank is not empty, the last track placed in the bank is taken out. The parameters that were stored with the track (x,y,z, ..., etc.) are reset and transport continues.

14. Return. After a source neutron history has been completed, all tally information is processed.

B. MCN Modifications Required for State Space Splitting

The major operations of state space splitting are performed by subroutines FEATEX and CLASS. Subroutine FEATEX (for an example, see Appendix A) is a user-supplied subroutine which contains the FORTRAN expressions for converting state space variables into a feature vector $\underline{F}$. Subroutine CLASS (see Appendix A) evaluates the discriminant functions $g_i = \underline{W}_i \cdot \underline{F}^*$ $(i = 1, \ldots, N_c - 1)$ and determines to which importance region or class $\underline{F}$ belongs.

A flow diagram of MCN with state space splitting is shown in Fig. III.2. The operations which have changed from the regular MCN code are enclosed in double boxes. The details of these changes are given below.

1. Calculate Source Parameters (Operation 1). After calculating the state space parameters (x,y,x,u,v,w,E,t), calls are made to subroutines FEATEX and CLASS to determine the source class C_S.

2. State Space Splitting at Collisions and at the Source (Operation 2). After a collision, a neutron's state space description has changed. Subroutines FEATEX and CLASS are called to determine the neutron's new classification C_N. This classification is compared to the classification C_o that the neutron had before the collision. The neutron undergoes splitting, Russian roulette, or is continued.

At the source, the class C_o does not exist. Instead the source class C_S is compared to a user-supplied class C_u to determine if splitting should take place.

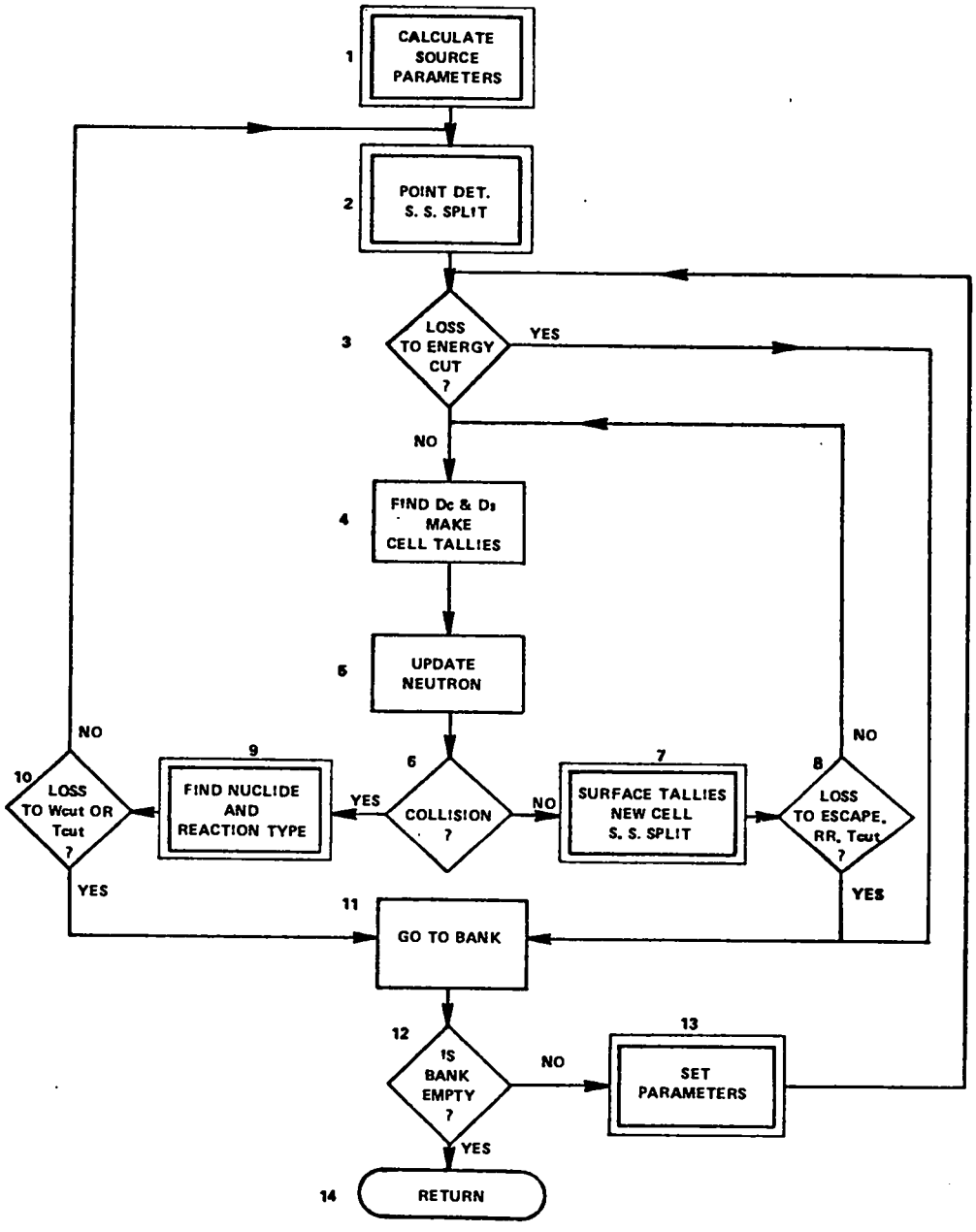

Fig. III.2. State Space Splitting

When a neutron is split, the classification of the neutron is banked along with the usual information. Energy splitting is eliminated.

3. State Space Splitting at Surface (Operation 7). When a neutron is transported to a surface, calls to FEATEX and CLASS are made to determine the neutron's new classification. The same logic that takes place after a collision is then used. Geometry splitting can be performed through the FEATEX and CLASS subroutines if desired.

4. State Space Splitting at Multiple Reactions (Operation 9). If a reaction occurs which produces additional neutrons ((n,xn) or fission), the additional neutrons are banked. Before banking, FEATEX and CLASS are called and the classification of the newly created neutron is compared with the class of the neutron which entered the reaction. The same logic that is used after collisions and at surfaces is then repeated here.

5. Setting Parameters (Operation 13). The classification of a neutron is withdrawn from the bank along with the usual information.

C. MCN Modifications Required for Pattern Recognition

The coding for adding pattern recognition is concerned with two functions: (1) prototype handling, and (2) the pattern classification algorithm. Fig. III.3 shows the MCN program logic after the addition of pattern recognition. Operations that must be changed to add pattern recognition to state space splitting are surrounded by a single solid frame. The double frame surrounds operations which must be changed when adding state space splitting to the regular MCN code. Note that to learn state space splitting surfaces requires a change in every nondecision operation. However, because no decision operation is altered, the structure of MCN remains the same.

1. Coding Required for Prototypes. Figure III.4 illustrates a typical history of a neutron in a Monte Carlo calculation. Each point between the directed line segments represents a collision or surface intersection. The circled points with more than one branch are points at which new tracks are created (by fission, (n,xn), or splitting). These new tracks are banked (stored) until the remaining track is terminated. In this particular example ten prototypes ($N_p = 10$) are created. When the track is terminated the total tally weight contributions of prototypes P_8, P_9, and P_{10} can be evaluated

$$T_8 = \frac{t_8 + t_9 + t_{10}}{wt_8} ,$$

$$T_9 = \frac{t_9 + t_{10}}{wt_9} , \text{ and}$$

$$T_{10} = \frac{t_{10}}{wt_{10}} ,$$

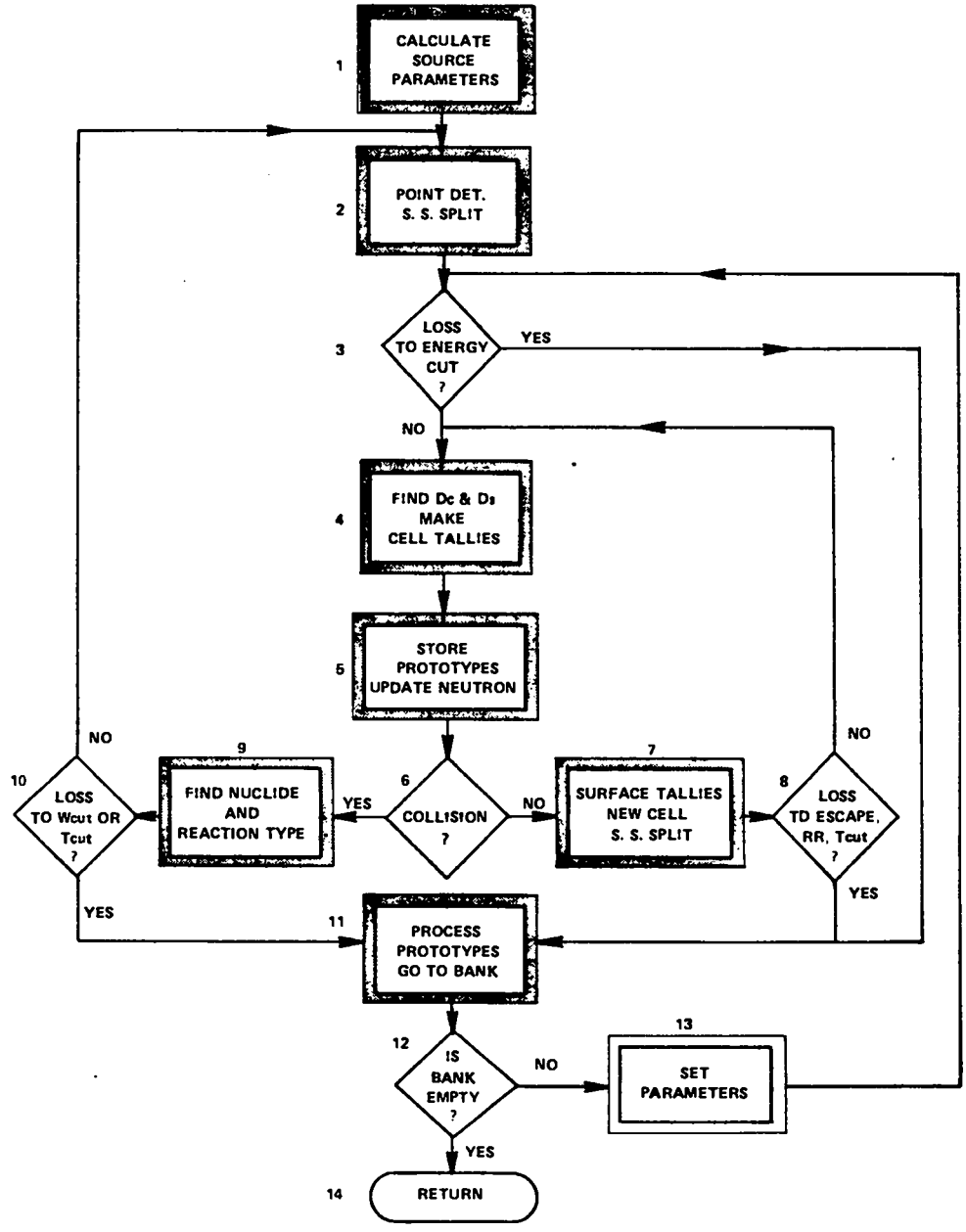

Fig. III.3. Pattern Recognition

where T, t, and wt are defined in Sec.II.B.1. Once this branch has been processed, prototypes P_8, P_9, and P_{10} can be forgotten; however, the total accumulated tally contribution ($t_8 + t_9 + t_{10}$) for the branch must be saved. Only if all three branches after prototype 7 have been processed can the prototypes prior to 8 be processed.

To keep track of this branching data and to allow for non-equal tally importances, the following data structure is used.

TWT(I) = a weight assigned to the Ith tally bin. This weight can be used to give tallies preferential treatment. A bin with a high weight relative to the others will have more particles directed towards it. Equivalent to D_b in Eq. (II.7).

PROTO(I) = a storage block used to save prototype feature vectors, weights, and tally weight contributions.

NPROTO = No. of prototypes stored in PROTO. Equivalent to N_p in Eq. (II.8).

NODS(I) = the prototype number where the Ith branch has occurred. For the example of Fig. III.4, NODS(1) = 1, NODS(2) = 5, and NODS(3) = 8. Note that the source track is counted as a branching intersection.

NNODS = No. of branching intersections. For the example, NNODS = 3.

NBNOD(I) = No. of branches remaining unprocessed on the Ith branching intersection . For the example, NBNOD(1) = 1, NBNOD(2) = 2, and NBNOD(3) = 2 (after P_8, P_9, and P_{10} are processed).

TAL(I) = the tally contribution accumulated at the Ith branching intersection. For the example, TAL(1) = 0, TAL(2) = 0, and TAL(3) = $t_8 + t_9 + t_{10}$.

IHCNOD(I)= the highest class (importance region) obtained by any prototype on the branches of the Ith branching intersection. Equivalent to C_H (Sec. II.B.1) for all branches of an intersection.

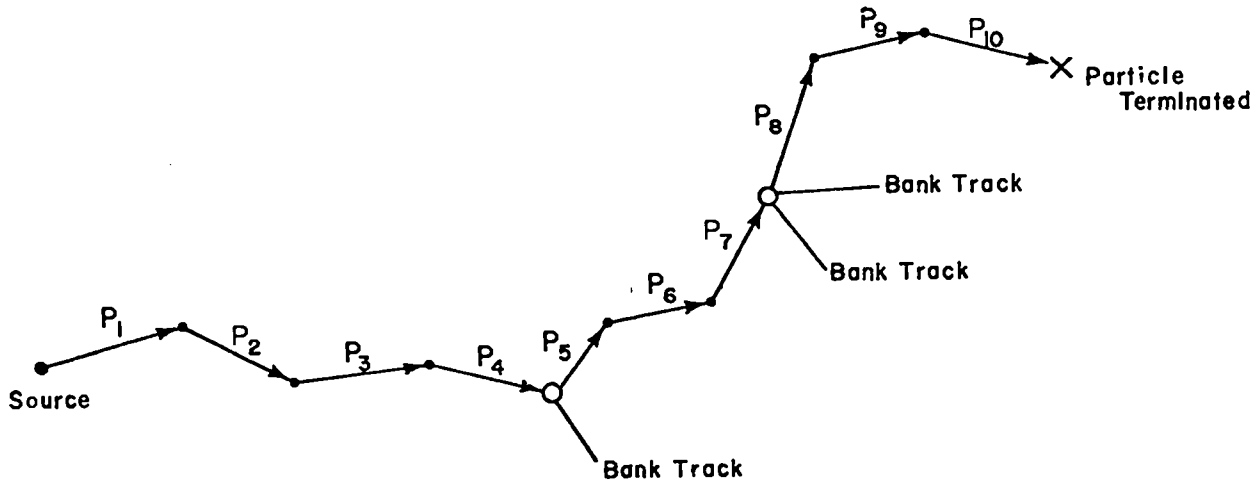

Fig. III.4. Branching Prototypes

Immediately preceding the updating of a particle (Operation 5, Fig. III.3) either at a surface intersection or at a collision, the following operations are performed:

N_w = NPROTO x (R + 2), where R = dimensionality of feature space

PROTO(N_w + I) = F(I), where F(I) is Ith component of the feature vector

PROTO(N_w + R + 1) = wt, where wt is the present weight of the particle

PROTO(N_w + R + 2) = t, where t is the present tally weight contribution to all cell tallies

NPROTO = NPROTO + 1

After surface tallies have been processed (Operation 7, Fig. III.3) and after point detector tallies have been processed (Operation 2, Fig. III.3), the tally contribution is updated as given by

$$\text{PROTO}(N_w + R + 2) = \text{PROTO}(N_w + R + 2) + t,$$

where t is the present tally weight contribution to all surface tallies or point detectors.

Whenever a track is created either by fission (n,xn) or splitting (Operations 2, 7, and 9 of Fig. III.3), the following operations are performed:

$$\begin{aligned} &\text{NNODS} = \text{NNODS} + 1 \\ &\text{NODS(NNODS)} = \text{NPROTO} \\ &\text{NBNOD(NNODS)} = N_t + 1 , \end{aligned}$$

where N_t is the number of additional tracks created.

When a neutron track has been killed, all prototypes created by that track are processed (Operation 11, Fig. III.3) before going to the bank for the next track. This processing is performed by subroutine PRPROT (see Appendix A). The number of prototypes processed at this time is given by

$$\text{JDIF} = \text{NPROTO} - \text{NODS(NNODS)} + \text{I}.$$

This processing begins with prototype NPROTO and ends with prototype NODS(NNODS). The accumulated tally weight contribution from the prototype being processed up to and including the prototype NPROTO is stored in variable BTAL. The variable TTAL is equivalent to T_j (Eq. (II.8)) and is given by

$$TTAL = BTAL/wt,$$

where wt is the weight of the neutron stored in the PROTO array. When a prototype is being processed, its feature vector $\underline{F}$ is extracted from the PROTO array and sent with TTAL to the pattern classification subroutine LEARN.

When all the prototypes on a branch have been processed, the branch is "trimmed" by the operations

$$NBNOD(NNODS) = NBNOD(NNODS) - 1$$
$$NPROTO = NPROTO - JDIF ,$$

and the branching intersection information is updated by the operations

$$TAL(NNODS) = TAL(NNODS) + BTAL$$
$$IHCNOD(NNODS) = \text{The maximum of } IHCNOD(NNODS) \text{ and } IHC.$$

The maximum class obtained on a branch, or the value of C_H (see Sec. II.B.1) for a branch, is given by IHC.

If no more branches exist at the NNODS intersection, processing is continued back to the (NNODS - 1) intersection and the NNODS intersection is trimmed.

2. Coding Required for Pattern Recognition. During processing, prototypes are sent to subroutine LEARN for adjustment of the splitting surface weight vectors. Subroutine LEARN (see Appendix A) has three major functions: (1) determine the classification of a prototype $\underline{F}$ according to the present splitting surface (i.e., the student classification = NS), (2) determine the classification as given by TTAL or the highest class obtained (i.e., the teacher classification = NT), and (3) make adjustments to $\underline{W}$ if NT $\neq$ NS.

The value of NS is determined by using the same logic as found in subroutine CLASS. Values of $g_i = \underline{W}_i \cdot \underline{Y}^*$ are evaluated starting with the uppermost surface until g_i is positive. The student class is given by NS = i + 1.

Determination of NT is more complicated as illustrated by Fig. III.5. The variable FLG is the surface number of the lowest surface used to separate prototypes for which $T_j \neq 0$ and is equivalent to $N_c - N_u$ (see Sec. II.B.1). If FLG = 0, surface $N_c - 1$ is used to separate non-zero T_j from zero T_j (i.e.,

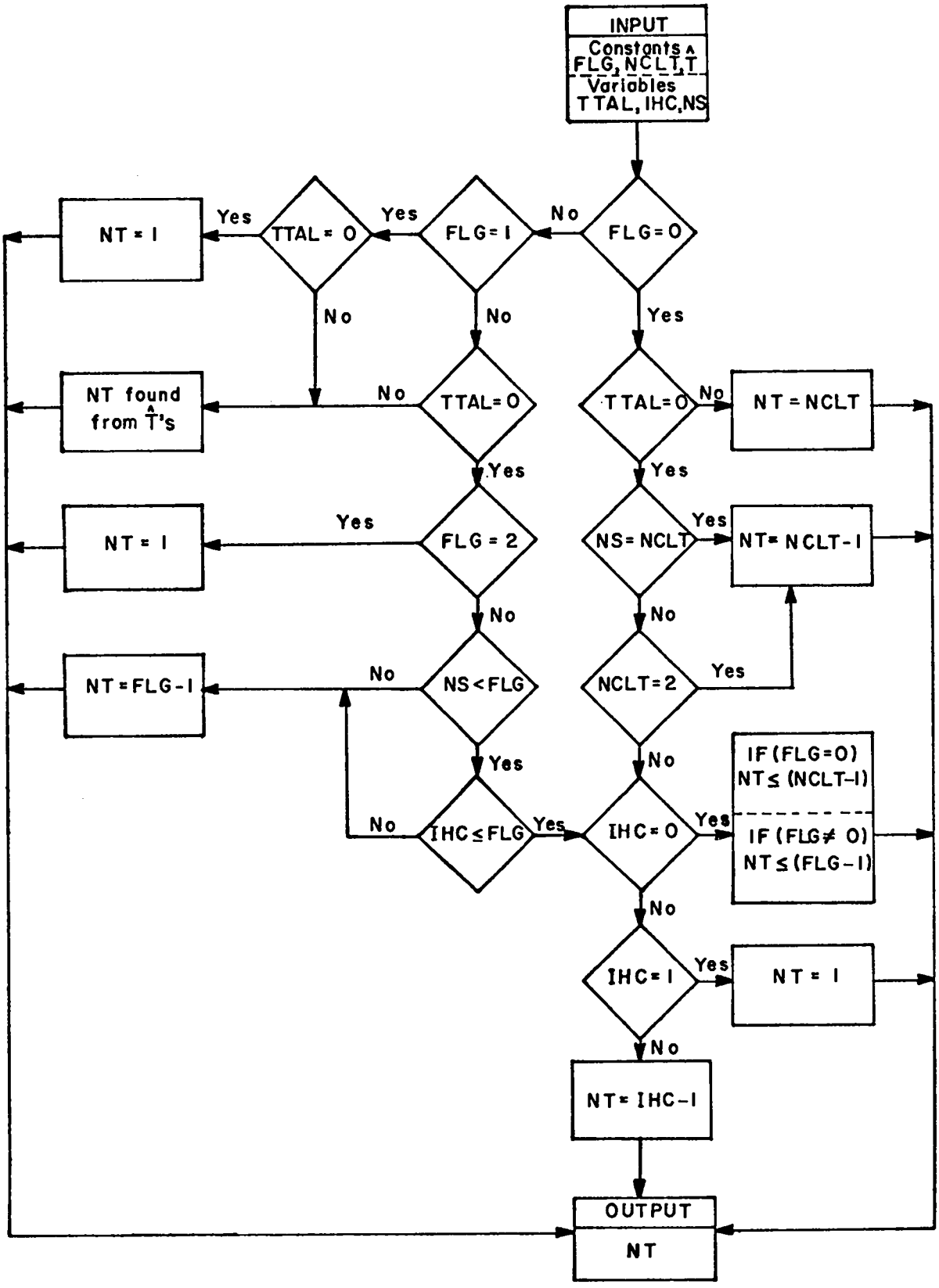

Fig. III.5. Determining the Teacher's Prototype Classification NT

$\hat{T}_{N_c-1} = 0$). If FLG $\neq$ 0, surface FLG - 1 separates non-zero T_j from zero T_j. The variables NCLT = N_c, FLG, and $\hat{T}$ are specified by the user or by default values.

If FLG = 0 and TTAL $\neq$ 0.0, the prototype is placed in class NCLT; if TTAL = 0, the class must be determined from the highest class obtained. The result of this procedure is that an F in class NCLT has a greater than 50% chance of contributing to a tally, an F in class (NCLT - 1) has a greater than 50% chance of leading to a prototype in class NCLT, an F in class (NCLT - 2) has a greater than 50% chance of leading to a prototype in class (NCLT - 1) or class NCLT, and so forth. An F in class 1 has a less than 50% chance of leading to a prototype outside of class 1.

If FLG $\neq$ 0, then surface (FLG - 1) is learned the same as surface (NCLT - 1) is when FLG = 0, surface (FLG - 2) is the same as (NCLT - 2), etc. Surfaces FLG through (NCLT - 1) are used to subdivide that region of feature space that has a greater than 50% chance of leading to a tally. Each one of the FLG through (NCLT - 1) surfaces has associated with it a $\hat{T}$ (see Sec. II.B.1) which determines the classification of the prototype. For example if

$$\begin{aligned} \text{NCLT} &= 5, \\ \text{FLG} &= 3, \\ \hat{T}_4 &= .5, \text{ and} \\ \hat{T}_3 &= .25, \end{aligned}$$

and a prototype was found to have TTAL = .21, the teacher class NT would be 3. A hypothetical division of two-dimensional feature space is shown in Fig. III.6 for the surfaces given above. The probability $P(\underline{F} \rightarrow \underline{F}' \geq 3)$ is the probability that an F will lead to an F′ in class 3 or greater.

The variables used to store information for splitting surface weight adjustment are

NPRO = total number of prototypes sent to pattern classifier since the calculation began,

NPR(K,L) = the number of prototypes whose feature vector components (f_1, f_2) fall between

$$\begin{aligned} (K - 1)/10 < f_1 < K/10 \\ (L - 1)/10 < f_2 < L/10 , \end{aligned}$$

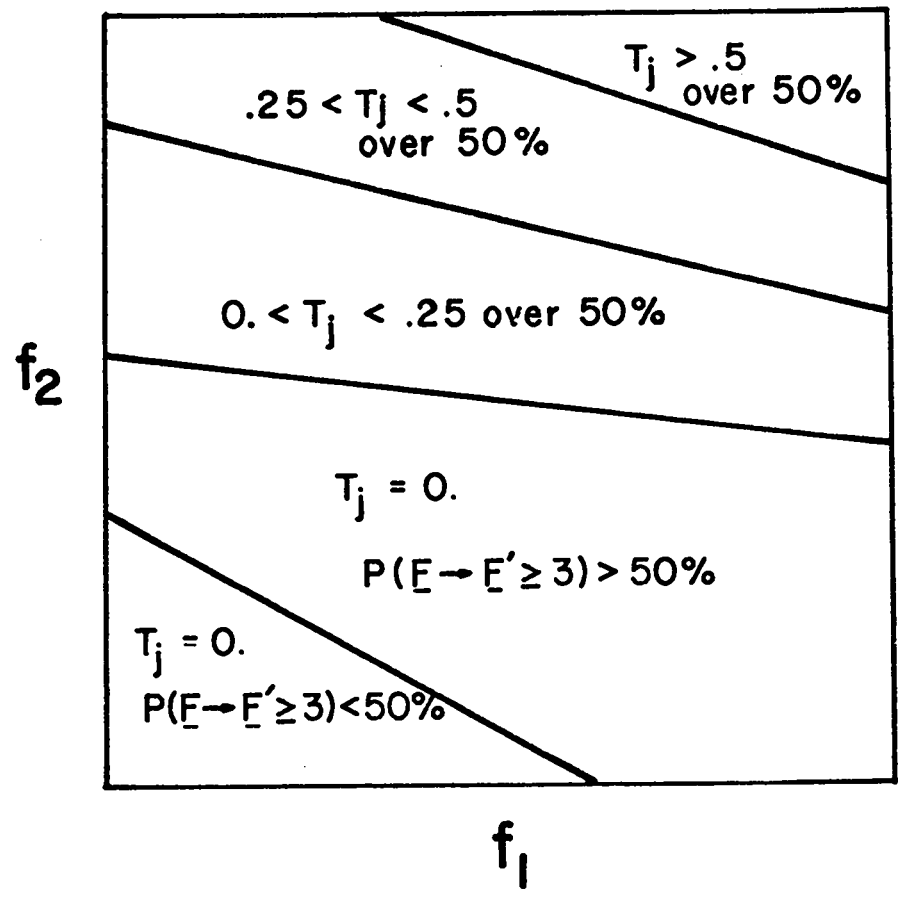

Fig. III.6. An Example of Splitting Surfaces

where K = 1, ..., 10
L = 1, ..., 10 .

If feature space is dimensioned greater than two, the dimensionality of NPR must be increased. This variable is used to estimate f_b (Eq. (II.12)).

TPR(M,I) = the I'th component for the $\underline{H}_J$ of the M'th splitting surface (H_J is defined in Sec. II.B.2)

RPR(M,I) = the I'th component for the $\underline{R}_J$ of the M'th splitting surface ($\underline{R}_j$ is defined in Sec. II.B.2)

F(I) = the I'th feature vector component of $\underline{F}^*$ for the current prototype
where I = 1, ..., R + 1
M = 1, ..., N_c - 1.

$$DIV = |\underline{F}^*| = \sqrt{F(1)^2 + F(2)^2 + \ldots + 1} \quad . \tag{III.1}$$

The variable DIV is equivalent to $1/f_a$ (Sec. II.B.2). Whenever NS > NT,

$$TPR(M,I) = TPR(M,I) - \frac{\lambda * NPRO * F(I)}{100 * NPR(K,L) * DIV}, \qquad \text{(III.2)}$$

where M = NT, ..., NS - 1

I = 1, ..., R + 1 ,

and whenever NT > NS,

$$TPR(M,I) = TPR(M,I) + \frac{\lambda * NPRO * F(I)}{100 * NPR(K,L) * DIV}, \qquad \text{(III.3)}$$

where M = NS, ..., NT - 1

I = 1, ..., R + 1.

In both cases,

$$RPR(M,I) = RPR(M,I) + TPR(M,I). \qquad \text{(III.4)}$$

When a source neutron's transport is complete, new average splitting surface locations are found from

$$ADFWT(M,I) = DFWT(M,I) + RPR(M,I)/NPRO , \qquad \text{(III.5)}$$

where ADFWT(M,I) = I'th component of the average weight vector $\overline{W}_J$ of the M'th splitting surface after NPRO prototypes.

DFWT(M,I) = I'th component of the initial weight vector $\underline{W}_o$ of the M'th splitting surfaces.

I = 1, ..., R + 1

M = 1, ..., N_c - 1 .

D. Control of Pattern Recognition Learning

Control of the pattern recognition system takes place after operation 14 (Fig. III.3) and consists of (1) setting the rate of learning of each splitting

surface, (2) determining when a surface has been learned, and (3) deciding when to use a surface for splitting.

The rate of learning is determined by the learning parameter λ (see Eq. (II.12)). The following variables are used when adjusting λ.

XAMDA(I) = λ for the I'th splitting surface

XFAC(J) J = 1, ..., 4 = constants used to set XAMDA

XMCLR(I) = the misclassification rate of the I'th splitting surface

XCLASA(I) = the number of prototypes that the teacher said were above surface I but the student disagreed

XCLASB(I) = the number of prototypes that the teacher said was below surface I but the student disagreed

ICONV = the number of prototypes that must be processed before the XAMDA(I) can be adjusted

IDCONV = the number of prototypes (since the last XAMDA(I) adjustment) that must be processed before XAMDA(I) can be adjusted again.

Whenever a prototype is misclassified XCLASA and XCLASB are incremented by

$$\frac{\text{NPRO}}{\text{NPR(K,L)}} \cdot 10^{-R} \quad ,$$

where (K,L) corresponds to the feature components (f_1,f_2). If $R \neq 2$, the NPR array must be increased or decreased. Thus the number of misclassified prototypes has been normalized by P(F) (see Sec. II.B.2). When the number of prototypes exceeds ICONV, the following operations are performed.

$$\begin{aligned} &\text{XMCLR(I)} = (\text{XCLAS(I)} + \text{XCLASB(I)})/\text{NPRO} \\ &\text{XAMDA(I)} = \text{XFAC(1)}*\text{XMCLR(I)} + \text{XFAC(2)} \\ &\text{if XAMDA(I)} > \text{XFAC(3) then XAMDA(I)} = \text{XFAC(3)} \\ &\text{if XAMDA(I)} < \text{XFAC(4) then XAMDA(I)} = \text{XFAC(4)} \\ &\text{for I} = 1, \ldots, N_c - 1. \end{aligned} \qquad \text{(III.6)}$$

NPRO is defined in the previous section. After the λ adjustments, ICONV is incremented by IDCONV and the calculation continues. The sign of XFAC(1) is negative which causes a surface with larger misclassification to be learned slowly. This is necessary since some of the misclassifications are due to overlapping and non-linearly separable classes, and not to actual misclassification. Typical values for XFAC are discussed in Sec. IV.

The following variables are used to terminate the learning of a surface:

ICONVT = the number of prototypes that must be processed before learning can be terminated

BOA(I) = XCLASB(I)/XCLASA(I) for surface I. BOA is a measure of how well a surface has been learned. When BOA = 1, the same number of prototypes are being misclassified on each side of the surface and the surface is considered to be learned.

CONV = the maximum deviation of BOA from 1 allowed for a surface to be learned.

After ICONV prototypes, the threshold ICONVT is checked. If NPRO > ICONVT then BOA(I) is calculated. If

$$1 - \text{CONV} < \text{BOA(I)} < 1 + \text{CONV},$$

then XAMDA(I) is set to zero. When XAMDA(I) is set to zero, the position of surface I is no longer altered. When all XAMDA(I) = 0 (I = 1, ..., N_c - 1), a flag is set in the code which terminates the collection and processing of prototypes. The code then performs state space splitting but no pattern recognition.

The decision of when to use the splitting sufaces for splitting is made by comparing the number of source particles NPS with a threshold NPSLRN. If NPS > NPSLRN, the surfaces are used.

Values for XAMDA, XFAC, ICONV, IDCONV, ICONVT, CONV, and NPSLRN are discussed in Sec. IV.

IV. PARAMETER STUDY

In the previous two sections, many parameters are described, but no quantitative values are given. It is the purpose of this section to describe briefly how the pattern recognition system reacts to variations in these parameters. This description cannot possibly cover the infinite possibilities that can arise in the general Monte Carlo problem. Instead only one very simple and non-time-consuming example is used. The parameters to be investigated are the following:

(1) initial location of the splitting surfaces and value of the learning parameter λ,

(2) values for the control parameters XFAC, ICONV, IDCONV, ICONVT, CONV, and

(3) selection of the number and type of splitting surfaces and NPSLRN.

A. The Sample Problem

The sample problem consists of the semi-infinite slab shown in Fig. IV.1. A neutron source exists at the surface x = 0 and is directed in the positive x direction. The tally consists of the current of neutrons, integrated over all time, crossing the surface at x = 10 cm.

The slab material has total and scattering macroscopic cross sections of .5 and .4 cm^{-1}. All scattering is isotropic in the laboratory system of coordinates and the neutron energy is unchanged by collisions. Because of the nature of this problem the state space variables y,z,v,w,E, and t are not important for splitting. Only the variables x and u are used in the transformation to feature space which is shown in Fig. IV.2 along with examples of three different kinds of splitting surfaces. Feature space is oriented so that the region above and to the right of a splitting surface is the most important region. This problem was chosen primarily because of its small computer time requirements. After starting 10 000 source neutrons and using less than a minute of computer time, the probability (= .037) of a neutron crossing the tally surface has a 3.5% relative error (1 σ). Such a "quick" problem is necessary for the numerous calculations used in this parameter study. The

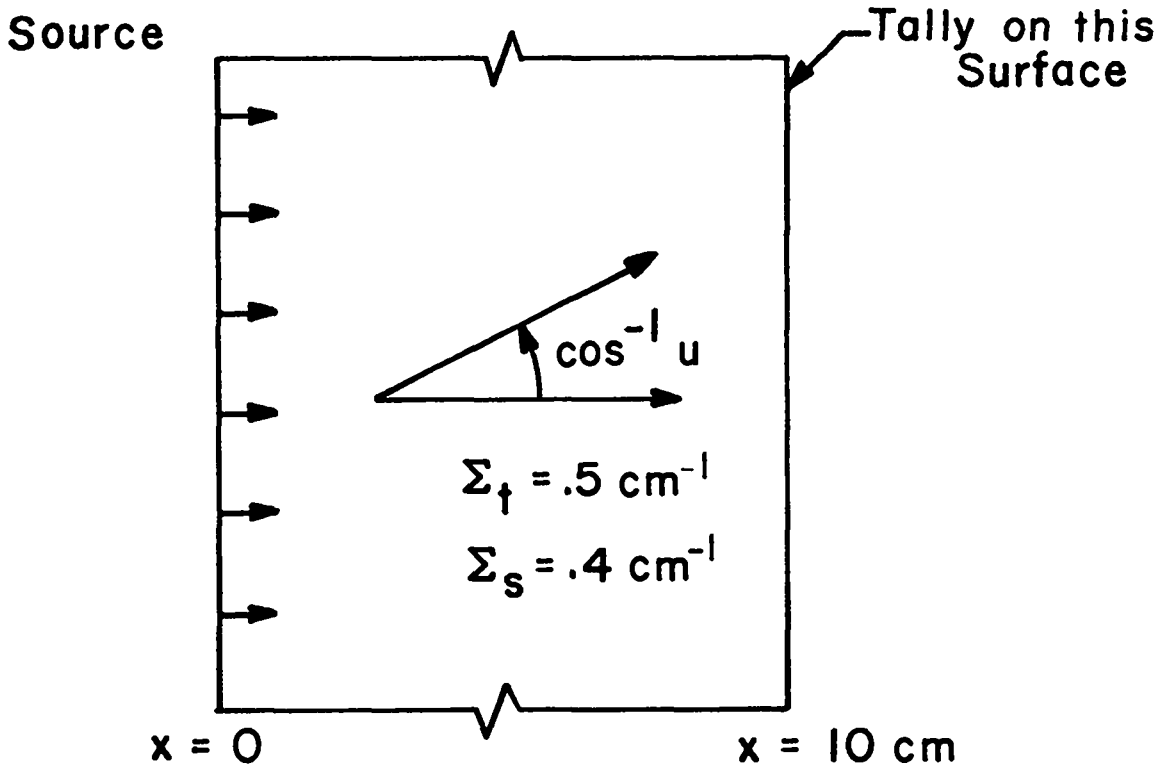

Fig. IV.1. The Sample Problem

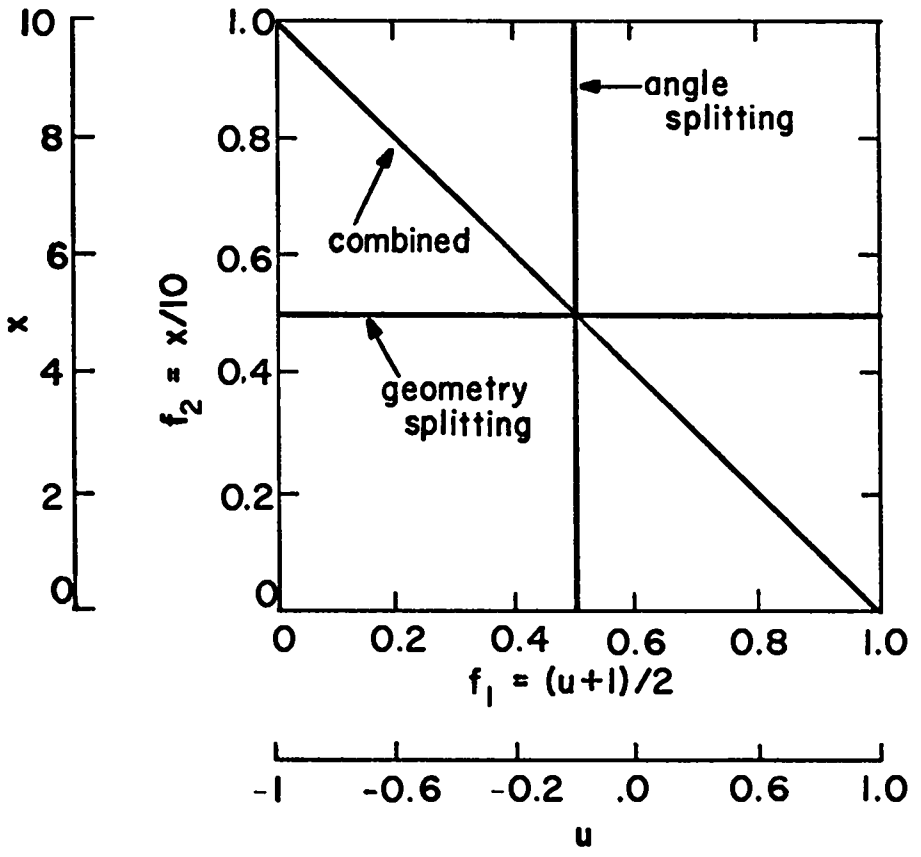

Fig. IV.2. Feature Space for the Sample Problem

probability distribution of prototypes $P(\underline{F})$ is estimated as described in Sec. II.B.2 and is given by

$$P(\underline{F}) \approx P_{m,n}(f_1|f_2) \cdot P_n(f_2) , \qquad \text{(IV.1)}$$

where $P_n(f_2)$ is the probability that a prototype is created with

$$.1(n - 1) \leq f_2 \leq .1n \qquad n = 1, \ldots, 10$$

and $P_{m,n}(f_1|f_2)$ is the probability that a prototype with an f_2 corresponding to m has an f_1 with

$$.1(m - 1) \leq f_1 \leq .1m \qquad m = 1, \ldots, 10.$$

Because scattering is isotropic in the sample problem all $P_{m,n}(f_1|f_2) = .1$ except in the source region (n = 1). At the source, neutrons are created at a

single point $(f_1,f_2) = (1,0)$. If R_s is the ratio of the number of source neutrons to the total number of prototypes, then $P_{m,1}(f_1|f_2)$ is given by

$$P_{m,1}(f_1|f_2) = .1\ (P_1(f_2) - R_s) \text{ for } m = 1, \ldots, 9 \text{ and}$$
$$P_{10,1}(f_1|f_2) = .1\ (P_1(f_2) - R_s) + R_s.$$

A plot of $P_n(f_2)$ is given in Fig. IV.3. Unlike the collision density, this distribution does not drop off exponentially because it does not depend on the neutron's weight. The $P_n(f_2)$ for the same problem but with a surface added at x = 2 cm (f_2 = .2) is also shown in Fig. IV.3. The result of this surface addition is to greatly increase the number of prototypes at f_2 = .2. The purpose of splitting is to alter the distribution by lowering P(F) for small $|\underline{F}|$ and increasing it for larger $|\underline{F}|$.

The tally weight contribution for this problem (see Eq. (II.7)) is given by

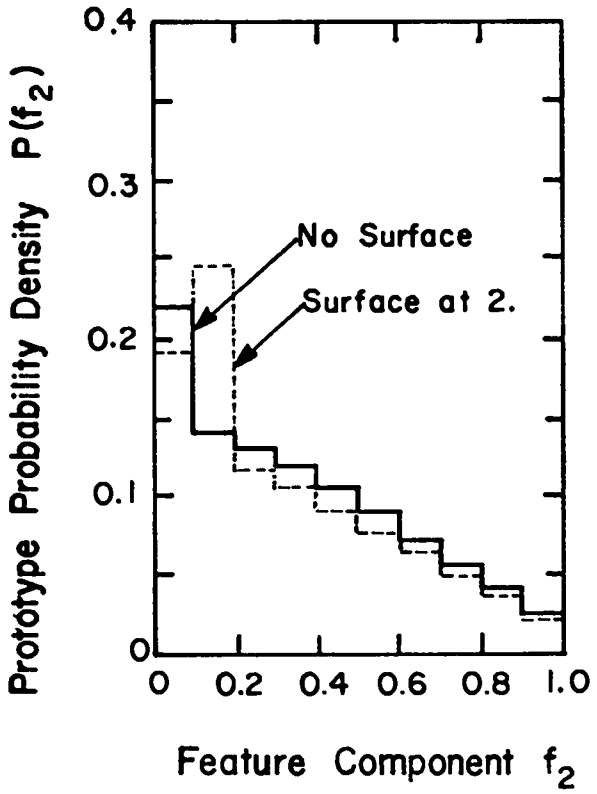

Fig. IV.3. Sample Problem Prototype Distribution

$$t_j = wt_j \, \delta_{1,j} \, ,$$

where for the single tally bin, $D_1 = 1$.

B. Effects of the Learning Parameter and Initial Weight Vector

The effect of varying λ (see Eq. II.12) and the initial weight vector $\underline{W}_o$ (see Eq. (II.11)) on the performance of the pattern recognition system cannot be determined in general. What is done in this section is to demonstrate how these parameters affect the sample problem described in the previous section.

Consider a single surface which separates feature space into two classes: (1) if $T_j = 0$, the prototype is said to belong to class 1; (2) if $T_j > 0$, the prototype belongs to class 2. This surface is similar to surface 2 in Fig. III.6. The surface is defined by a weight vector $\underline{W}$, originally given by $\underline{W}_o$. Since feature space is two-dimensional ($R = 2$), $\underline{W}_o$ is given by

$$\underline{W}_o = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

The discriminant function (see Eq. (II.4)) is given by

$$g(\underline{F}) = w_1 f_1 + w_2 f_2 + w_3 .$$

The splitting surface is defined by $g(\underline{F}) = 0$ and is given by

$$\begin{aligned} & w_1 f_1 + w_2 f_2 + w_3 = 0, \text{ or} \\ & \frac{w_1}{w_3} f_1 + \frac{w_2}{w_3} f_2 + 1 = 0. \end{aligned} \qquad \text{(IV.2)}$$

Because of the required orientation of feature space (see Sec. II.A.2), w_1 and w_2 must be positive and w_3 must be negative. In this research, the initial value of w_3 is always set to -1 which reduces Eq. (IV.2) to

$$w_1 f_1 + w_2 f_2 - 1 = 0.$$

The initial value of $\underline{W}_o$ depends on two values, w_1 and w_2. The ratio w_2/w_1 determines the slope of the splitting surface in feature space. A "logical guess" for w_2/w_1 is

$$w_2/w_1 = 1.$$

The remaining single parameter to set is $w_1 = w_2$. Three choices of w_1 are shown in Fig. IV.4. These three initial $\underline{W}_o$'s were used with $\lambda = 10^{-3}$. The splitting surfaces learned after 10000 source neutrons are shown in Fig. IV.4. A plot of BOA (see Sec. III.D) versus the number of source neutrons is shown in Fig. IV.5 for these three runs and for three runs using $\lambda = 10^{-4}$. As can be seen from these plots, the value of λ has much more effect on the convergence rate than does the initial $\underline{W}_o$. Runs made with $\lambda = 10^{-2}$ result in the surface learning being unstable.

Similar calculations were performed for a surface for which $\hat{T} = .5$. This type of surface is similar to surface 4 in Fig. III.6. The results are similar to those shown in Fig. IV.4 For this surface, convergence is too slow when $\lambda = 10^{-3}$. With $\lambda = 10^{-2}$, the convergence is very rapid, yet still stable. With $\lambda = 10^{-1}$, the surface learning is unstable.

Before an optimum λ can be found, a measure of instability must be defined. The following scheme is used in this research. During the learning of a surface, BOA should converge to 1 monotonically when approached from above (see Fig. IV.5). The measure of instability used is the number of times that BOA increases during this initial approach to one. A plot of instability versus λ is shown in Fig. IV.6 for the surface learned in Fig. IV.5 using $w_1 = 2.0$. Also shown in Fig. IV.6 is the number of source neutrons (NPS) required before BOA < 1.1. When calculating the instability shown in Fig. IV.6, BOA is checked after every 200 source particles for 10 000 source particles thus producing 50 data points. If the behavior of BOA were purely random, the instability would be 25. A λ is desired which minimizes convergence time but does not cause the learning to be unstable. For this surface $\lambda \approx 10^{-3}$ is suitable. For the $\hat{T} = .5$ surface, the value of λ selected was 8×10^{-3}. A third surface was learned similar to surface 1 of Fig. III.6. For this surface the optimum λ was also $\approx 10^{-3}$. These results are sumarized below.

Surface Type (Fig. III.6)	Misclassification Rate XMCLR	Learning Parameter λ
4	.14	8×10^{-3}
2	.30	1×10^{-3}
1	.20	1×10^{-3}

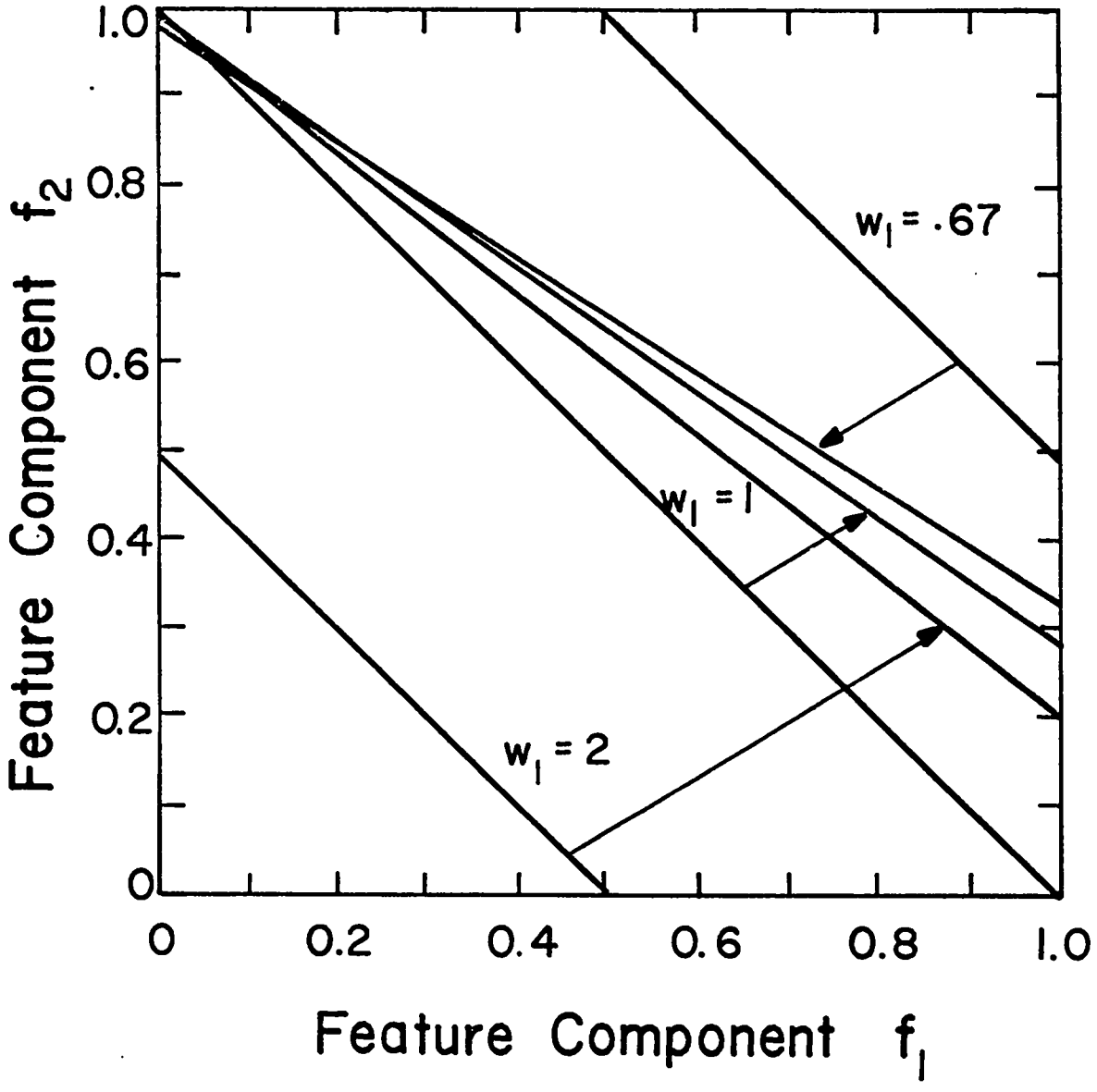

Fig. IV.4. Initial and Final Splitting Surface Locations

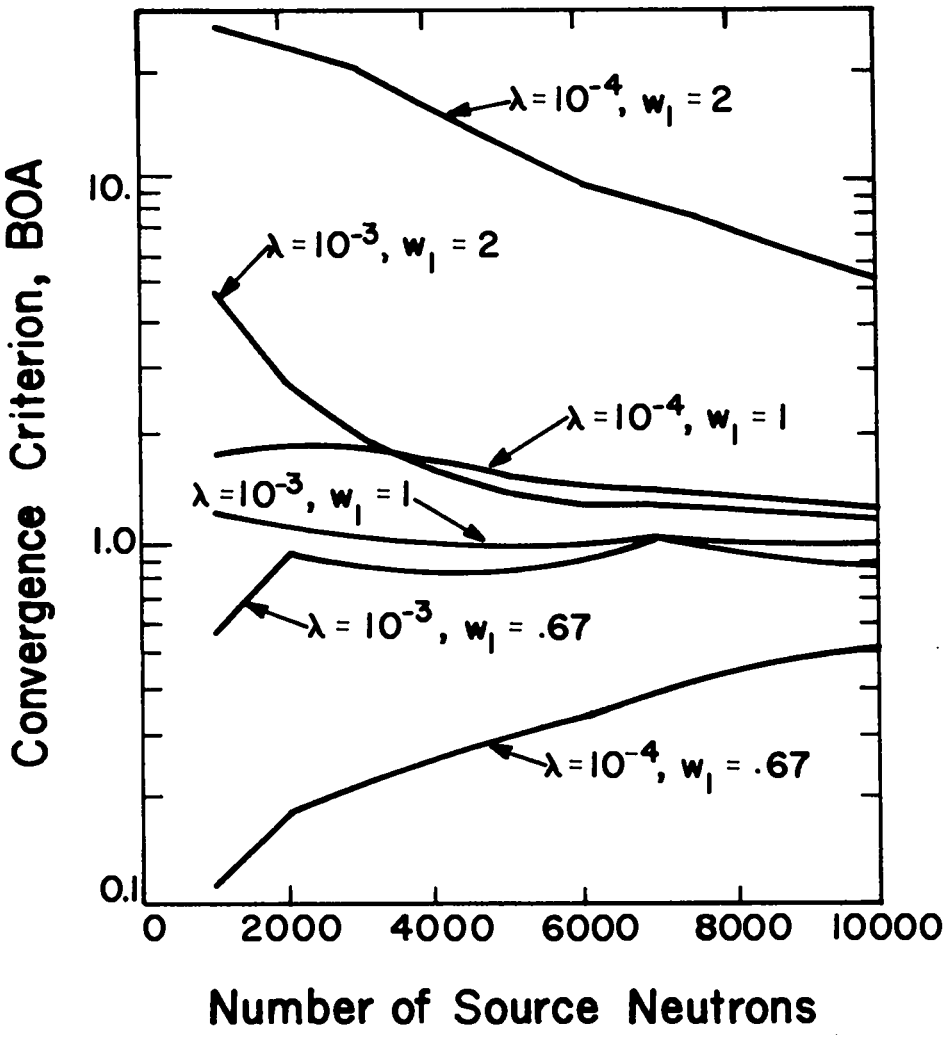

Fig. IV.5. Initial Condition and Learning Parameter Effects on Convergence Rate

Using the above data and Eq. (III.7) results in the following recommended values

$$\begin{aligned} XFAC(1) &= -.0434 \\ XFAC(2) &= .014076 \\ XFAC(3) &= .05 \\ XFAC(4) &= 1 \times 10^{-3}. \end{aligned} \qquad \text{(IV.3)}$$

Although this heuristic approach to a variable λ is not optimal, it does relieve the user of specifying a λ for each surface. Whether the above constants for λ are generally applicable will be seen in Sec. V.

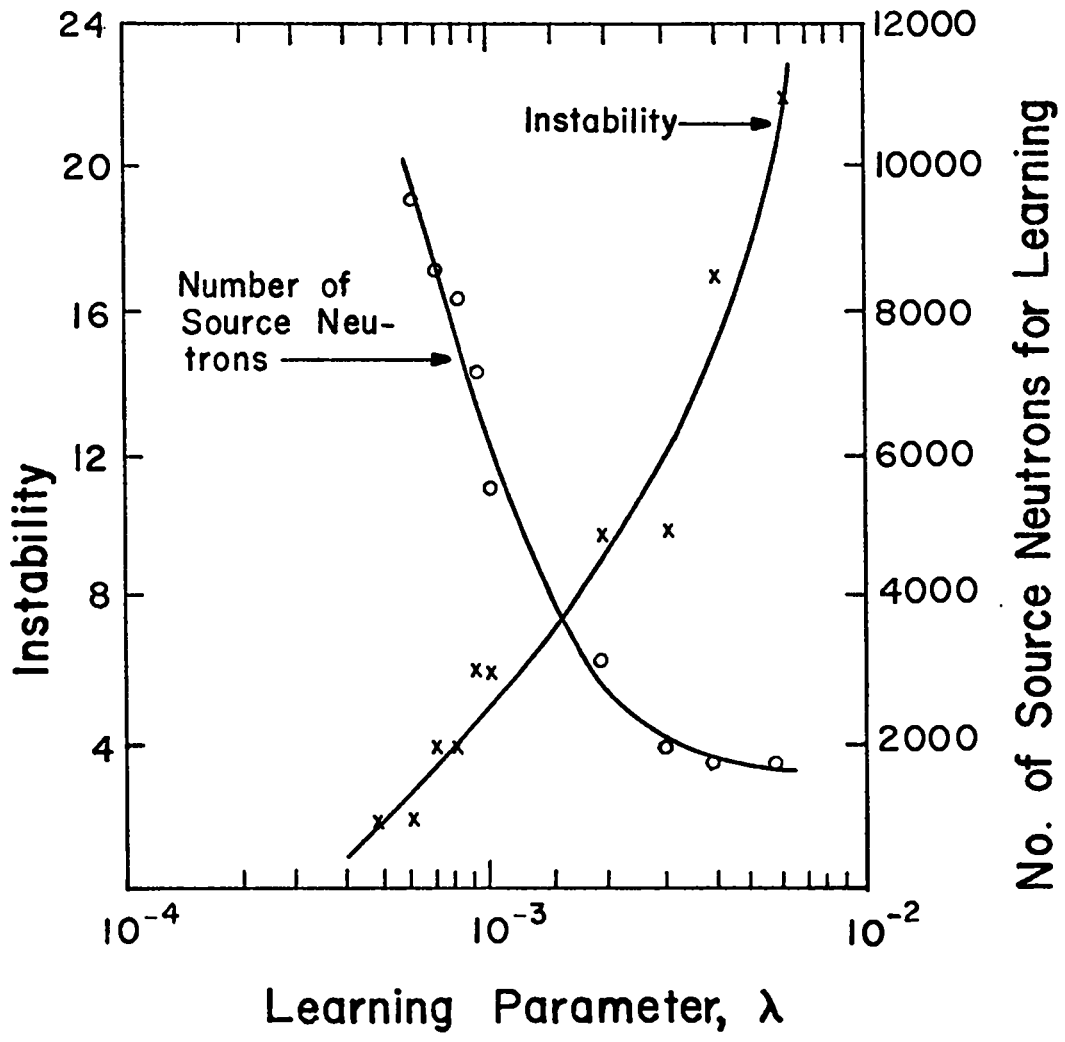

Fig. IV.6. Selecting a Suitable Learning Parameter

In summary, Eq. (III.7) is used to select a λ using the values given in Eq. (IV.3). The (R + 1)'st weight vector component for the initial weight vector is always set to -1. A reasonable "guess" to use for the initial surface location is

$$w_1 = w_2 = \ldots = w_R = c_i \ ,$$

where c_i is a constant for the i'th splitting surface. These constants are selected so as to distribute splitting surfaces uniformly across feature space. For example, if there are three surfaces, suitable c_i's would be

$$c_1 = 2,$$

$$c_2 = 1, \text{ and}$$

$$c_3 = .67.$$

Of course, if the user has a priori information concerning optimal surface locations, this information should be used to reduce learning time.

C. Control Parameters

The value of XMCLR (see Eq. (III.6)) must be learned during the calculation before it can be used in Eq. (III.7). An initial λ is assigned until ICONV prototypes have been processed. The effect of varying ICONV is shown in Fig. IV.7 for the surface learned in the previous section using $\hat{T} = .5$ and an initial $\underline{W}_o$ = [.75 .75 -1]. Initially, λ is set to 10^{-3}. This value is chosen since it is safe to use for most surfaces (i.e., it is small enough to be stable). For

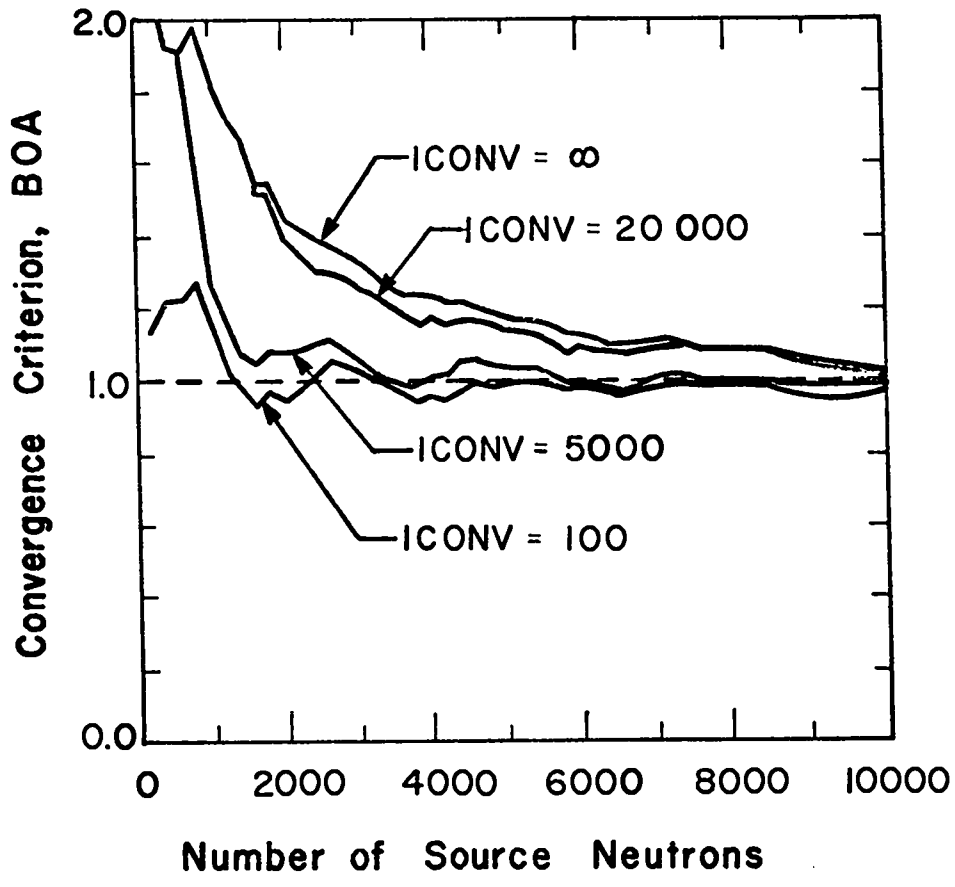

Fig. IV.7. Effect of ICONV on Convergence Rate

these four runs, IDCONV = 500. As seen from the figure, the ICONV = 100 run appears to be the most desirable. However, this depends on how good the estimate of XMCLR after 100 prototypes is to the true value. Because of this, it would be dangerous to use ICONV = 100 in a general problem. The curve for which ICONV = 5000 drops rapidly after 5000 prototypes to the ICONV = 100 curve. The difference in the two curves is caused by the bias in BOA due to the first 5000 prototypes. When ICONV = 20 000, the convergence is improved very little from the ICONV = ∞ curve.

All runs were found to be relatively insensitive to the value of IDCONV. This is because XMCLR is well known after ICONV prototypes. With too small an IDCONV, computer time is wasted. If too large, XMCLR may not be represented accurately. An IDCONV anywhere from a few 100 to a few 1000 was found to be sufficient.

As seen from Fig. IV.7, there will always be some fluctuation of BOA even after convergence. These fluctuations are caused by statistical deviations in the prototypes and exist even with a fixed splitting surface. The fluctuations are usually less than 5% but can be greater. Because of this, the value of CONV should be a minimum of .05. The value of ICONVT should be large enough to prevent a "false convergence" from being used. An example of a false convergence is shown by the ICONV = 100 curve of Fig. IV.7. This problem is similar to the problem encountered when trying to stop a Monte Carlo calculation when the relative error is below a threshold. In this sample problem, a value of ICONVT = 20 000 was found to be safe, provided the surface learning is stable. Using ICONVT = 20 000, the CONV parameter was varied resulting in the following:

CONV	Source Neutrons Required for Learning
.05	4336
.10	2870
.15	1569

In all three cases, the surface was not located where it would be if the run were continued indefinitely. As the number of source neutrons increases, the movement in surface location becomes small compared to the initial changes. Thus although a small CONV does result in a "more optimal" location, the improvement is small compared to the additional source neutrons required.

In summary, for the problem considered, the following parameters are found to be satisfactory:

ICONV = 5000,
IDCONV = 1000,
ICONVT = 20000, and
CONV = .10.

D. Splitting Surface Selection and NPSLRN

The user is required to specify the number of surfaces ($N_c - 1$) and the type of surfaces (FLG, see Sec. III.C.2) that he wishes to use. There are two types of surfaces: (1) those for which $\hat{T}_i = 0$ and those for which $\hat{T}_i > 0$ ($i = 1, \ldots, N_c - 1$). The value of FLG is the lowest surface number for which $\hat{T}_i > 0$. Thus surfaces i = FLG, FLG + 1, ..., ($N_c - 1$) have $\hat{T}_i > 0$. It follows that surfaces i = 1, ..., (FLG - 1) have $\hat{T}_i = 0$. If FLG = 0, all surfaces have $\hat{T}_i = 0$. For the sample problem described earlier, three surfaces are investigated in this section:

Surface	Description
C	$\hat{T}_i = .5$, i = 1,2,or 3
B	$\hat{T}_i = 0.$, i = 1 or 2
A	$\hat{T}_1 = 0.$, below A $P(\underline{F} \rightarrow \underline{F}' \geq 3) < 50\%$, above A $P(\underline{F} \rightarrow \underline{F}' \geq 3) > 50\%$.

Note that surface A cannot be used unless surface B is specified since its definition depends on surface B. Thus with these three surfaces, five combinations are possible:

Run	Surface	FLG	$N_c - 1$
1	C	1	1
2	B	0	1
3	C and B	2	2
4	B and A	0	2
5	A,B, and C	3	3

To determine which of these surface combinations is best, the surfaces were first learned using the parameters recommended in the previous two sections. The learned surface locations are:

Surface	w_1	w_2
C	1.10	1.48
B	.82	.88
A	.61	.80

These locations were then used in the above five runs using state space splitting only (i.e., the surface locations were not modified). The results are given below relative to run 4 (see Sec. II.C for a definition of the terms).

Run	Relative σ_s^2	Relative t_p	Relative Cost
1	2.16	.53	1.13
2	2.10	.59	1.15
3	1.88	.69	1.30
4	1.0	1.0	1.0
5	.93	1.19	1.11

As expected, the variance decreases as the number of surfaces increases and the computer time per neutron increases.

The above data does not include the computer time required to learn the splitting surface. To include this cost, an NPSLRN (see Sec. III.D) must be selected. The maximum value of NPSLRN is the value at which the splitting surfaces have been learned and all λ's = 0. The minimum value possible is zero. With NPSLRN = 0, the splitting surfaces are used for splitting throughout the calculation. This is desirable if the initial "guess" for the surface location is a good one. However, splitting with surfaces from a poor guess could waste computer time on unproductive splitting.

Using run 4 with learning and three different values of NPSLRN results in the following:

NPSLRN	Cost (Relative)
2000	1.05
1000	1.00
0	1.00

The above values are relative to the previous run 4 made without learning. All runs were performed for 30 000 source neutrons. If splitting is not used during learning, 1943 neutrons are required to learn the surfaces. Figure IV.8 illustrates computer time and variance (relative to run 4) as a function of the number of source neutrons for the NPSLRN = 2000 run. After the surfaces are learned (1943 neutrons) but before they are used (2000 neutrons), the relative computer time drops below one because neither splitting nor learning is taking place. During the learning of splitting surfaces, about 8% more computer time is used. Averaged over the entire run, this amount of time is negligible. The sample variance without splitting is approximately 70% higher than with. After splitting starts, the variance is given by

$$\sigma_s^2 = \frac{N_{wo}\,(\sigma_{wo}^2 - \sigma_w^2) + N\sigma_w^2}{N}\,, \qquad \text{(IV.4)}$$

where σ_{wo}^2 = sample variance of the calculation if no splitting is used

σ_w^2 = sample variance if splitting is used throughout the calculation

N_{wo} = number of source neutrons started while the surface is being learned, but before splitting is started

N = total number of neutrons started.

Equation (IV.4) can be used provided no splitting takes place until learning is complete. If this is not the case, σ_w^2 must be replaced by $\overline{\sigma}_w^2$ where $\overline{\sigma}_w^2$ is the variance obtained from splitting after NPSLRN source neutrons. If the number of neutrons during which learning and splitting both take place is small compared to the number for which splitting takes place without learning, then $\overline{\sigma}_w^2 \approx \sigma_w^2$. As is seen from the data, the value of NPSLRN has little effect on the computer costs provided it is small compared to the total number of neutrons started (30 000 for these calculations). A value of 1 000 is suitable since most Monte Carlo runs are on the order of 10 000 or greater.

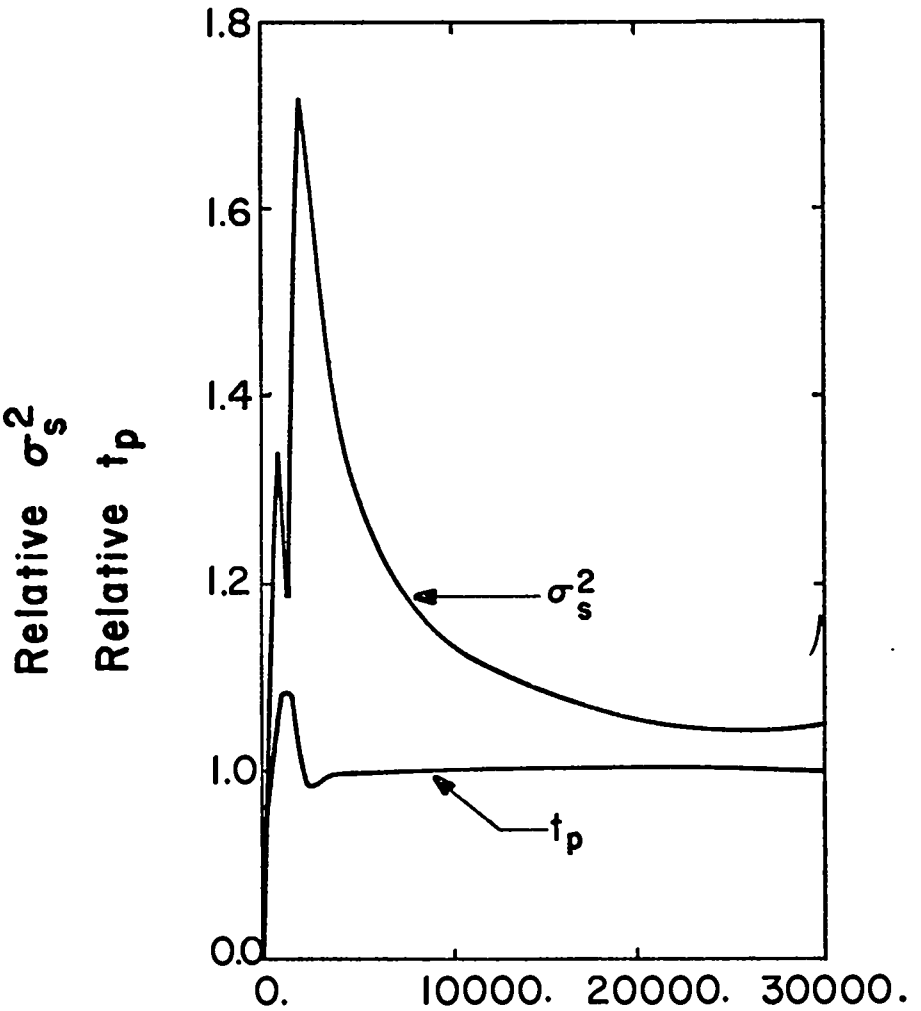

Fig. IV.8. Relative Computer Time and Variance

In summary, the selection of the number and type of splitting surfaces is a nontrivial problem. For the sample problem used in this section, FLG = 0 and $(N_c - 1) = 2$ appear to be the best values. However, this selection will undoubtedly be problem dependent.

E. Summary

Of the parameters described in this report, only two must be selected by the user.

(1) $(N_c - 1)$, the number of splitting surfaces, and

(2) FLG, the type of splitting surfaces.

All other parameters can be assigned default values. If a user wishes to supply his own values instead of the defaults, he may. In fact, for such parameters as

the initial surface locations, the user can save computer time if he provides values better than the default. In addition, the user must provide the feature space description and the tally whose variance is to be reduced.

V. SAMPLE PROBLEMS

It is the purpose of this section to demonstrate how the previously described technique can be used to solve a variety of Monte Carlo problems. There is no attempt to try to represent all classes of Monte Carlo problems. Instead, the geometry of a single Monte Carlo problem is used (Sec. V.A). For this geometry, several different tallies are investigated that require different feature space approaches. Each tally is investigated in an independent calculation. The reason for this approach is that it allows a single description of a geometry for several different feature space examples. If it was desired to solve all tallies in a single run, one would use the feature space configuration for the tally with the highest relative error.

In all problems, the parameter values chosen in the previous section are used. The user must select feature space functions, the number of splitting surfaces, the type of surface, and the tally bins whose variances are to be reduced. All surfaces are initially located uniformly across feature space.

A. The Geometry

The geometry and material compositions used for the example problems are those of a double ring thermal neutron coincidence counter.[12] This problem was selected because: (1) it has tallies which are infrequent enough to require splitting; (2) neutron energies cover the full range from a few MeV to thermal; (3) time response is important; (4) the geometry is reasonably complex; (5) the problem is not excessively expensive to run yet it is not trivial (10 to 25 minutes on a CDC 7600 computer).

The geometry is illustrated in Fig. V.1 and the material compositions are given in Table V.1. The PuO_2 sample consists of several isotopes of Pu. The spontaneous fission of ^{240}Pu provides the source of neutrons. Four tallies are investigated:

(1) the current of neutrons leaving the outside surface of the outer region of CH_2,

(2) the current of neutrons leaving the top and bottom surfaces of the outer region of CH_2,

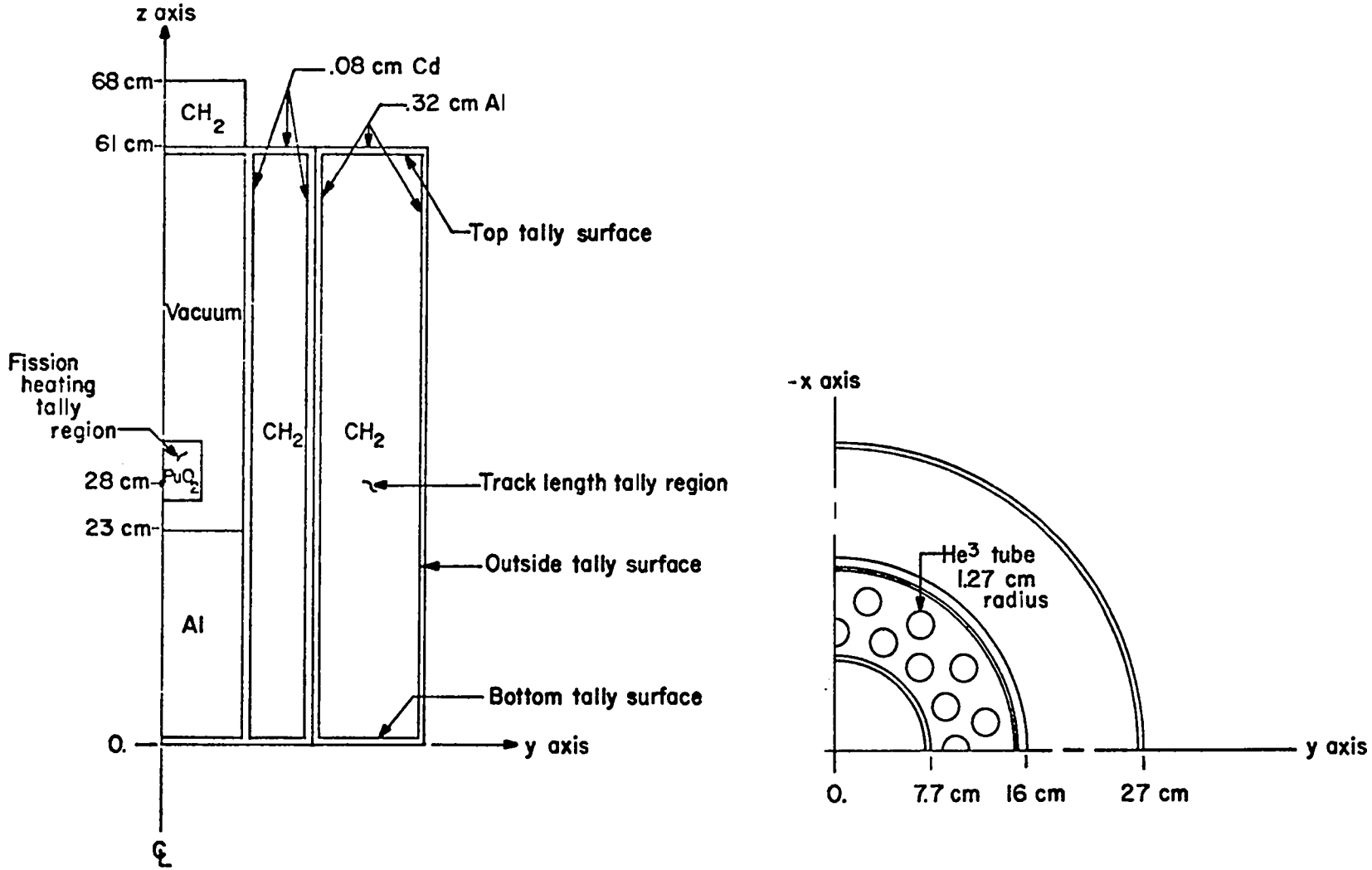

Fig. V.1. Sample Problem Geometry

TABLE V.1

MATERIAL COMPOSITIONS

Material	g/cm^3	$(Atoms/cm^3)x10^{-24}$
PuO_2	1.5	$1.00x10^{-2}$
Al	2.7	$6.03x10^{-2}$
CH_2	.91	$1.17x10^{-1}$
Cd	8.64	$4.63x10^{-2}$
He^3	$5.4x10^{-4}$	$1.07x10^{-4}$

(3) the heating due to induced fissions in the PuO_2 (a track length estimate) and,

(4) the flux (track length) of thermal neutrons as a function of time in the outside region of CH_2.

Tallies 1 and 2 are integrated over all outward directions. Tallies 1, 2, and 3 are divided into four energy bins:

(1) 0 to 4×10^{-7} MeV,

(2) 4×10^{-7} to 1×10^{-3} MeV,

(3) 1×10^{-3} to 1 MeV, and

(4) 1 to 20 MeV.

Tallies 1, 2, and 3 are integrated over all time whereas tally 4 is divided into 10 time bins of .1 ms width from 1 to 2 ms. Tally 4 has one energy bin from 0 to 4×10^{-7} MeV.

B. Distance Splitting

One of the most common features found in Monte Carlo problems is distance. In distance splitting, the feature consists of the distance from the particle's present position to the desired tally region. Geometry splitting is a form of distance splitting where the geometric regions closest to the tally have higher importances than those farther away. In this section, two examples of distance splitting are presented using tallies 1 and 2.

Without splitting, the largest relative error for tally 1 is 9.8% for the second energy bin after 10 000 neutron histories. For tally 2, the largest error is 25% for the second energy bin of the lower surface after 20 000

histories. Since these maximum errors determine the running time of the calculation, they will be used for computer cost comparisons.

1. Tally 1. The distance d from a point (x,y,z) to the tally surface is given by

$$d = 26.68 - \sqrt{x^2 + y^2} \quad . \tag{V.1}$$

For this example, splitting will be limited to the outside CH_2 region, i.e., $0 \leq d \leq 10.68$. After normalizing to 1 and orienting the feature so that the importance increases as the feature increases (see Sec. II.A.2), the resulting feature is given by

$$f_1 = \frac{10.68-d}{10.68} \quad . \tag{V.2}$$

If $f_1 < 0$ or $f_1 > 1$, the feature vector is not used as a prototype and the neutron is treated as if it were in the class of lowest importance (class 1). This capability of excluding selected regions of state space from splitting and learning results in more efficient sampling.

It should be noted that for this example, state space splitting is equivalent to geometry splitting. However, if one used geometry splitting, he would have to select splitting surface locations and subdivide the outer CH_2 into a number of smaller regions.

Having selected the feature, the next step is to select the tally bins whose variances are to be reduced, the number of splitting surfaces, and the type of surfaces. The selection of tallies is done by setting the variable TWT (see Sec. III.C.1). For this problem the TWT for the four energy bins of the outside surface tally are set to 1. All other TWT are set to zero. This problem was run with $(N_c - 1) = 1$, 2 and FLG = 0, 1. The results obtained after 10 000 histories are presented in Table V.2. All data are for the second energy group and are relative to the calculation made without state space splitting. Although the variance was reduced from 29 to 56%, the computer cost was reduced from only 11 to 35%. The increased computer time was due primarily to the increase in the number of tracks.

For the $(N_c - 1) = 2$, FLG = 1 case, the upper surface (separating class 2 from 3) converged to $f_1 = .727$ and was learned after 1 000 histories (58 000

TABLE V.2

RESULTS FOR SAMPLE PROBLEMS

Tally	Feature	$(N_c - 1)$	FLG	σ_s^2	Relative t_p	$\sigma_s^2 t_p$
1	Distance	2	0	.44	2.02	.89
		2	1	.52	1.25	.65
		1	0	.65	1.34	.87
		1	1	.71	1.11	.79
2	Distance	2	0	.35	1.08	.37
		3	0	.18	1.31	.24
2	Energy	2	0	.74	1.10	.81
		3	0	.64	1.16	.74
3	Direction	2	0	.48	1.09	.53
4	Time	2	0	.38	1.36	.52
2	Distance & Energy	3	0	.27	1.10	.29
4	Direction & Energy	2	0	.46	1.08	.50

prototypes). The lower surface converged to .716 after 1 320 histories (84 000 prototypes).

2. Tally 2. In this example, it is desired to direct neutrons in two different directions. The feature used is

$$f_1 = \frac{|Z - 30.5|}{15.09} - 1 \ . \qquad (V.3)$$

Like the previous example, splitting is allowed only in the outer CH_2 region. From Eq. (V.3), if $15.41 < Z < 45.59$, $f_1 < 0$. Thus splitting and prototype production take place only when a neutron is within 15.09 cm from a tally surface. The TWT for all energy bins for top and bottom tally surfaces were set

to 1. Because very few neutrons arrive at the tally, only FLG = 0 surfaces are used. The results after 20 000 histories using N_c = 3 and 4 are given in Table V.2. These values are relative to those obtained for the second energy bin of the lower surface without splitting.

For $(N_c - 1) = 3$, the upper two surfaces (separating class 2 from 3 and class 3 from 4) were learned after 1 100 histories (20 000 prototypes). Their final locations were at $f_1 = .5$ and $f_1 = .73$. The lower surface was not learned after 20 000 histories (656 000 prototypes) and was located at $f_1 = .04$.

C. Energy Splitting

In this example, it is desired to reduce the variance of the second energy bin of tally 2. Thus TWT = 1 for the second bin and TWT = 0 for the others. As in the previous two examples, splitting and prototype production are limited to the outer CH_2 region. In this region the feature is given by

$$f_1 = \frac{-(\log_{10}\left(\frac{E}{4}\right) + 2)}{5} . \qquad (V.4)$$

Eq. (V.4) further restricts splitting to neutrons with energies from 4×10^{-2} to 4×10^{-7} MeV. A neutron with an energy outside this range is placed in class 1.

Results after 20 000 histories are presented in Table V.2 for FLG = 0 and $(N_c - 1) = 2$ and 3. As can be seen, for this tally, energy splitting appears to be far less effective than distance splitting. Thus the selection of features greatly influences the performance of state space splitting.

The middle splitting surface for $(N_c - 1) = 3$ was learned after 6 000 histories (62 000 prototypes) and had a final location of $f_1 = .86$. The lowest surface (between classes 1 and 2) was unlearned after 20 000 histories (208 000 prototypes) at which time BOA(1) = 0. It was located at $f_1 = .08$. The highest surface was also unlearned after 20 000 histories with BOA(3) = 7.6. Its final location was $f_1 = .99$.

D. Direction Splitting

Next to distance features, direction features are probably the most common. In direction splitting, it is desired to produce more neutrons that are traveling towards the tally region. The variable used is the cosine μ of the angle between the present neutron direction (u,v,w) and the direction to the desired tally (u_T, v_T, w_T).

Tally 3 was used for this example. The distance from (x,y,z) to a point location at the center of the PuO_2 is given by

$$d = \sqrt{x^2 + y^2 + (z - 28)^2} \ . \qquad (V.5)$$

The value of μ and the feature is given by

$$f_1 = \mu = -u \left(\frac{x}{d}\right) - v \left(\frac{y}{d}\right) - w \left(\frac{z-28}{d}\right) \ . \qquad (V.6)$$

Splitting is limited to the region where $15 < z < 60$ and $\sqrt{x^2 + y^2} < 8.2$. If $f_1 < 0$, the neutron is placed in class 1.

The results obtained after 10 000 histories using $(N_c - 1) = 2$ and FLG = 0 are presented in Table V.2. These values are relative to a calculation made without splitting and are for the second energy bin. Without splitting, the relative error obtained is 14% after 10 000 histories.

Neither surface was learned after 10 000 histories (30 000 prototypes). The values of BOA(I) were 6 and > 10 at this point indicating that few particles contribute to the tally even in the most important region of feature space. The final surface locations were at $f_1 = .76$ and $f_1 = .99$.

E. Time Splitting

Tally 4 was used for this example. The relative errors obtained without splitting after 10 000 histories are presented in Table V.3. In practice one would split so as to create more neutrons past 1.6 ms. However, even with splitting, this would take extremely long computer runs. For demonstration purposes, it was decided to reduce the 1.3-1.4 ms bin. Therefore, the TWT for this bin was set to 1, all others zero.

Splitting was restricted to the outer CH_2 region. Within this region the feature is given by

$$f_1 = t/1.4 \ , \qquad (V.7)$$

where t is time in ms. If $f_1 > 1$, the neutron is placed in class 1.

The cost reduction obtained for the 1.3-1.4 ms bin with $(N_c - 1) = 2$, and FLG = 0 and 10 000 histories is given in Table V.2. The effect on the

TABLE V.3

RESULTS FOR TIME SPLITTING

Time Bin (ms)	Relative Error Without Splitting	With Splitting
1.0-1.1	14%	8.3%
1.1-1.2	16%	10%
1.2-1.3	18%	12%
1.3-1.4	21%	15%
1.4-1.5	29%	26%
1.5-1.6	51%	36%
1.6-1.7	No contributions	50%
all others	No contributions	No contributions

variance of the other bins is seen from Table V.3. Although only a single bin was specified for cost reduction, the variances of all tally bins were reduced.

Both surface locations were learned within 500 histories (30 000 prototypes). The final surface locations were at $f_1 = .45$ and $f_1 = .73$.

F. Two-Dimensional Feature Space

In the previous examples, the feature vector F consists of a single component f_1. The only additional input required for multidimensional feature space is the additional feature components.

1. Distance and Energy Splitting. All input (except for the feature specifications) is identical to the example described in Sec. V.B.2 using $(N_c - 1) = 3$. Feature component f_1 is given by Eq. (V.3), f_2 by Eq. (V.4). Splitting is restricted to the outer CH_2 region.

Results after 20 000 histories are given in Table V.2. This feature space configuration is not as effective as the distance feature alone for the same number of surfaces. The initial and final surface locations are shown in Fig. V.2. After 20 000 histories (41 000 prototypes), none of the surfaces was learned.

2. Direction and Energy Splitting. All input (except for feature specifications) is identical to the example described in Sec. V.D using $(N_c - 1) = 2$. Feature components f_1 and f_2 are given by Eqs. (V.6) and (V.4), respectively. Splitting was restricted as described in Secs. V.D and V.C.

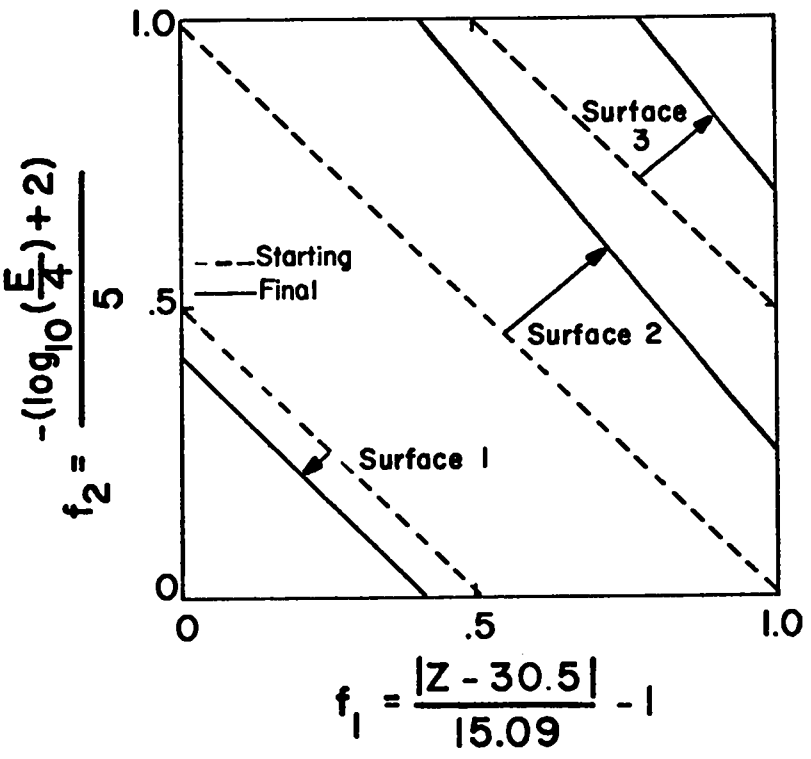

Fig. V.2. Splitting Surface Locations for Distance and Energy Splitting

Results after 10 000 histories are presented in Table V.2. The cost is slightly less than direction splitting alone. After 10 000 histories (12 146 prototypes), neither surface was learned. Final locations are shown in Fig. V.3.

G. Summary

Several conclusions are made below concerning computer costs, surface learning, and surface selection. These conclusions are based on the experience obtained from the previous examples.

1. Computer Cost Reduction. From Eq. (II.16), the relative computer cost can be expressed as

$$R_{cost} = R_\sigma \cdot R_t \qquad (V.8)$$

$$R_\sigma = \frac{\left(\sigma_s^2\right)_w}{\left(\sigma_s^2\right)_{wo}}$$

$$R_t = \frac{\left(t_p\right)_w}{\left(t_p\right)_{wo}} .$$

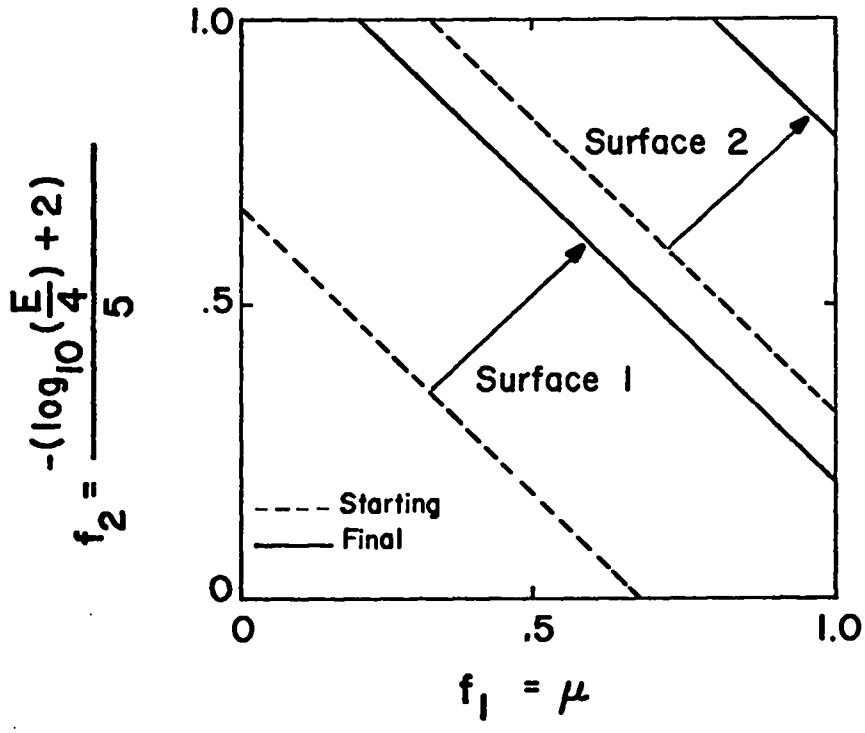

Fig. V.3. Splitting Surface Locations for Direction and Energy Splitting

The variance factor R_σ is given by

$$R_\sigma = R_{ss} \cdot R_{pr} \quad , \tag{V.9}$$

where R_{ss} is the R_σ that would be obtained using state space splitting alone and R_{pr} is the additional reduction obtained because surfaces are located by pattern recognition. The value of R_{ss} must be determined for the average user, a study beyond the scope of this research. For the sample problems considered, $.18 < R_\sigma < .74$.

The computer time factor R_t is given by

$$R_t = R_{SP} \cdot R_L \quad , \tag{V.10}$$

where R_{SP} is the ratio of computer times for particle transport only and R_L is given by

$$R_L = \frac{N_L t_L + N_T t_T}{N_T t_T} \quad , \tag{V.11}$$

where N_T = total number of histories

N_L = number of histories required for surface learning

t_L = computer time per history required for learning

t_T = computer time per history required for transport.

For cases in which $N_L = N_T$, $R_L \sim 1.1$. As $N_T >> N_L$, R_L approaches 1. The value of R_{SP} varies depending on the relative amount of splitting and Russian roulette taking place. If more splitting occurs, $R_{SP} > 1$, if more roulette, $R_{SP} < 1$. For the sample problems considered, $1.08 \leq R_t \leq 2.02$.

The resultant value of R_{cost} ranged from .89 to .24. Table V.4 presents the best R_{cost} obtained for each tally. The relative errors obtained without splitting after 10 000 histories are also listed. The data indicate that the slower the calculations, the greater the improvement.

When comparing the estimated variance of two calculations, the reliability of these estimates is important. In this research, the variances were observed throughout the calculation. Only after the sample variance had become relatively constant, were the variances used. The estimated mean obtained with splitting was within 1σ of the mean obtained without splitting for all runs compared.

2. Surface Learning. Of the 25 surfaces summarized in Table V.2, 14 of the surfaces were learned before completion of the Monte Carlo calculation (10 000 or 20 000 histories).

Of the 11 unlearned surfaces, 6 had converged but the BOA criterion (see Sec. IV.C) was not satisfied (i.e., either BOA > 1.1 or BOA < .9). These 6

TABLE V.4

COST REDUCTION VS PROBLEM DIFFICULTY

Best R_{cost}	Relative Error (10 000 Histories)	Tally
.65	9.8%	1
.53	14%	3
.50	21%	4
.24	33%	2

are summarized in Table V.5. The surfaces are of two types - those for which BOA > 1.1 and those for which BOA = 0. When BOA > 1, the surface location approaches $\underline{F}$ = [1 1 ... 1]. If feature space is constructed so that no region (or an infinitesimally small region about [1 1 ... 1]) satisfies the conditions for class N_c, then BOA cannot converge to 1. For the cases for which BOA = 0, no prototypes belonged to class 1 (according to the teacher classification, NT). In this situation, surface 1 approaches [0 0 ... 0] at a rate proportional to the production rate of prototypes below the surface. In both of these cases, the control system should recognize the situation and stop the learning of the surfaces involved.

The number of histories required for a surface to be learned depends on the number of prototypes produced per history, P_h. The value of P_h varied from 1.2 to 107 for the sample problems. The remaining 5 unlearned surfaces were for runs in which P_h = 1.2 and 2.1. The surfaces that were learned required from 373 to 13 000 histories (20 000 to 250 000 prototypes). Prototypes are created at surface intersections and collisions. Thus problems in which the number of collisions and intersections per history is large should have rapid learning.

3. Selection of the Type and Number of Splitting Surfaces. When the majority of feature space contains prototypes which contribute to a tally greater than 50% of the time, the important region should be subdivided by setting FLG $\neq$ 0. In the previous examples, this was done only for tally 1 since there were few contributions for the other tallies. In most cases when FLG $\neq$ 0 is required, the Monte Carlo problem probably does not require splitting. An

TABLE V.5

CONVERGED BUT UNLEARNED SURFACES

Tally	No. of Surfaces (N_c - 1)	Surface No.	Number of Prototypes	BOA	Surface Location
2	2	2	121 887	2.03	.99
2	3	3	207 837	1.58	.99
3	2	2	29 728	> 10	.99
2	3	1	655 612	0.0	.039
2	3	1	207 837	0.0	.082
2	3	1	41 169	0.0	.37/.37

exception is the point detector tally for which all collision points yield a contribution. Because of this, FLG = 0 should be used as a default for all tallies except the point detector.

The number of surfaces selected is dependent on the type of problem being solved. For the sample problems used in this research, 1 to 3 surfaces were used. If for surface 1, BOA = 0, additional surfaces would be of little value. This criterion should be used to allow the controller to select the number of surfaces. This could be done by starting with a large number of surfaces and eliminating all except one of those for which BOA = 0.

In summary, both the surface type specification (FLG) and the number of surfaces should be set by the control system unless otherwise desired by the user.

VI. CONCLUSIONS AND RECOMMENDATIONS

A pattern recognition system has been implemented in the MCN computer code and used successfully to learn state space splitting surface locations. For the sample problems considered, computer cost was reduced from 11% to 75%. As with conventional geometry splitting, the more difficult the problem, the greater the savings.

Required user input consists of

(1) the feature vector description,

(2) the tally bins whose variances are to be reduced, and

(3) the number and type of splitting surfaces.

The number and type of surfaces could be selected automatically if desired. The user is relieved of much of the quantitative input usually required by variance reduction schemes.

The general applicability of the technique depends somewhat on the cleverness of the user in selecting a good feature space configuration. Further studies are required to determine how optimal features can be selected.

There are many areas of possible improvements: (1) non-linear splitting surfaces, (2) a more efficient pattern recognition algorithm, (3) multiple feature space configurations for a single problem, (4) a better learning criterion, etc. Although such improvements are desirable, they are not required before the technique can be used. It is recommended that the technique be used by general users before any improvements are made in order to determine any user-oriented problems that may exist.

ACKNOWLEDGMENT

This study was funded by the Los Alamos Scientific Laboratory's New Research Initiatives Program.

REFERENCES

1. N. J. Nilsson, Learning Machines (McGraw-Hill Book Co., New York, 1965).

2. H. C. Andrews, Introduction to Mathematical Techniques in Pattern Recognition (Wiley-Interscience, New York, 1972).

3. J. M. Hammersley and D. C. Handscomb, Monte Carlo Methods (John Wiley & Sons, Inc., New York, 1964).

4. M. Clark and K. F. Hansen, Numerical Methods of Reactor Analysis (Academic Press, New York, 1964).

5. L. L. Carter and E. D. Cashwell, Particle-Transport Simulation with the Monte Carlo Method, ERDA Critical Review Series, US Energy Research and Development Administration TID 26607.

6. J. L. Macdonald, "Investigation of Pattern Recognition Techniques for the Identification of Splitting Surfaces in Monte Carlo Transport Calculations," thesis, University of Texas at Austin; also Los Alamos Scientific Laboratory report LA-6015-T (August 1975).

7. E. D. Cashwell, J. R. Neergard, W. M. Taylor, and G. D. Turner, "MCN: A Neutron Monte Carlo Code," Los Alamos Scientific Laboratory report LA-4751 (1972).

8. T. Y. Young and T. W. Calvert, Classification, Estimation, and Pattern Recognition (American Elsevier Publishing Co., New York, 1974).

9. K. S. Fu, Sequential Methods in Pattern Recognition and Machine Learning (Academic Press, New York, 1968).

10. K. Fukunaga, Introduction to Statistical Pattern Recognition (Academic Press, New York, 1972).

11. L. L. Carter, E. D. Cashwell, C. J. Everett, et al., "Monte Carlo Code Development in Los Alamos," Monte Carlo Conf., Argonne National Laboratory, Argonne, Illinois, July 1-3, 1974; also Los Alamos Scientific Laboratory report LA-5903-MS (March 1975).

12. N. Baron, "Studies of Thermal-Neutron Coincidence-Counting Techniques," in "Nuclear Safeguards Research and Development Program Status Report, January--April 1977," Los Alamos Scientific Laboratory report LA-6849-PR, (August 1977) p. 39.

APPENDIX A

FORTRAN SUBROUTINE LISTINGS

```
      SUBROUTINE PRPROT
C
C     SUBROUTINE PRPROT PROCESSES THE PROTOTYPES BY DECIPHERING
C     THE PROTO ARRAY AND SENDING PROTOTYPES TO SUBROUTINE LEARN
C     ONE PROTOTYPR AT A TIME
C
      BTAL=0.0
      IHC=0
C
C     JDIF  =  NO. OF PROTOTYPES TO BE PROCESSED
 59   JDIF=NPROTO-NODS(NNODS)+1
C
C     SEND PROTOTYPES TO SUBROUTINE LEARN
      DO 60 J=1,JDIF
C     JM  =  PROTOTYPE BEING PROCESSED
      JM=NPROTO+1-J
      NPNW=JM*(NWT+1)
C     BTAL  =  TALLY OF THIS BRANCH OF PROTOTYPES
C     TTAL = IMPORTANCE OF PROTOTYPE
C     PROTO(NPNW)  =  TALLY OF PROTOTYPE BEING PROCESSED
C     PROTO(NPNW-1) = WEIGHT OF PROTOTYPE
      BTAL=BTAL+PROTO(NPNW)
      TTAL=BTAL/PROTO(NPNW-1)
      CALL LEARN(TTAL,IHC,PROTO(NPNW-NWT))
 60   CONTINUE
C
C     A BRANCH HAS BEEN TRIMMED
      NBNOD(NNODS)=NBNOD(NNODS)-1
      IHCNOD(NNODS)=MAX0(IHCNOD(NNODS),IHC)
C     TAL(NNODS)  =  TALLY ACCUMULATED AT THE NNODS INTERSECTION
      TAL(NNODS)=TAL(NNODS)+BTAL
      NPROTO=NPROTO-JDIF
      IF(NBNOD(NNODS).EQ.0) 15,14
C
C     NO MORE BRANCHES TO PROCESS ON NNODS BRANCH
 15   CONTINUE
      BTAL=TAL(NNODS)
      IHC=IHCNOD(NNODS)
      TAL(NNODS)=0.0
      IHCNOD(NNODS)=0
```

```
      NNODS=NNODS-1
      IF(NNODS.GT.0) GO TO 59
C
 14   CONTINUE
      NPRSV=NPROTO
      IF(NPRSV.EQ.0) NPRSV=1
C
C
      RETURN
      END
      SUBROUTINE CLASS

C     FEAT(I)     =ITH FEATURE VECTOR COMPONENT
C     ADFWT(I,J) = JTH WEIGHT OF DISCRIMINANT FUNCTION FOR CLASS I,I+1
C     NWT         = NUMBER OF WEIGHTS = NUMBER OF FEATURES + 1
C     NCLM1       = NUMBER OF CLASSES - 1
C     NCLC        = CURRENT CLASS
C     NCLO        = PREVIOUS CLASS
C     NFI = NUMBER OF FEATURES THAT CHANGE AT INTERSECTIONS

      DIMENSION G2(10)

      ENTRY CLASS2
      IF(MPRFLG.EQ.2) GO TO 71
      IF(FEAT(1).EQ.-1.) GO TO 71
      DO 10 K=1,NCLM1
      I=NCLT-K
      G=ADFWT(I,NWT)
      DO 20 J=1,NWTM1
 20   G=G+ADFWT(I,J)*FEAT(J)
      IF(G) 10,10,30
 10   CONTINUE
      I=0
 30   NCLC=I+1
      RETURN
   71 NCLC=FIO(IA)
      RETURN
      END
      SUBROUTINE FEATEX
C
C     FEATURES USED FOR TWO DIMENSIONAL SLAB EXAMPLE
C
```

```
      ENTRY FEATEX2
      FEAT(1)=(U+1.)/2.
      FEAT(2)=X/SRC(1)
      RETURN
      END
      SUBROUTINE LEARN(T,IHC,XX)
C
C     THIS SUBROUTINE ADJUSTS THE DISCRIMINANT FUNCTIONS
C     IF A PROTOTYPE HAS BEEN MISCLASSIFIED
C
      DIMENSION XX(MNOFT),G(MNOSF)
      DATA BSFN/MNOSF*0./
      DATA ASFN/MNOSF*0./
      DATA XCLASA/MNOSF*0./
      DATA XCLASB/MNOSF*0./
      DATA NPLOST/0/,NPOOR/0/
      DATA NPRO/0/
      DATA BINFO/MNOSFH*0.0/,NPR/100*0/
C
C     SEE IF PROTOTYPE IS WITHIN RANGE
      IF(XX(1).EQ.-1.) GO TO 403
C
C     FIND CLASS ACCORDING TO STUDENT = NS
      DO 10 K=1,NCLM1
      I=NCLT-K
      G(I)=ADFWT(I,NWT)
      DO 20 J=1,NWTM1
 20   G(I)=G(I)+ADFWT(I,J)*XX(J)
      IF(G(I)) 10,10,30
 10   CONTINUE
      I=0
 30   NS=I+1
C
C     FIND CLASS ACCORDING TO TEACHER = NT
      IF(IFRFLG-1) 1000,900,800
 800  IF(T.EQ.0.) 801,250
 801  IF(IFRFLG.EQ.2) GO TO 901
      IF(NS.LT.IFRFLG.AND.IHC.LE.IFRFLG) GO TO 220
      NT=IFRFLG-1
      GO TO 300
 900  IF(T.NE.0.) GO TO 250
 901  NT=1
      GO TO 300
```

```
C     USE THIS SECTION FOR SELDOM OCCURING TALLIES
 1000 IF(T.EQ.0) GO TO 200
      NT=NCLT
      GO TO 300
  200 IF(NS.NE.NCLT.AND.NCLT.NE.2) GO TO 220
      NT=NCLM1
      GO TO 300
  220 IF(IHC.EQ.0) GO TO 260
      IF(IHC.NE.1) GO TO 240
      NT=1
      GO TO 300
  240 NT=IHC-1
      GO TO 300
C     USE THIS SECTION FOR FREQUENT TALLIES
  250 DO 255 NT=IFRFLG,NCLT
      IF(T.LE.XIMP(2,NT)) 251,255
  251 IF(T.GE.XIMP(1,NT)) 300,402
  255 CONTINUE
      GO TO 402
C     LOST PROTOTYPE
  260 NS=IFRFLG-1
      IF(IFRFLG.EQ.0) NS=NCLM1
      NT=NS
      IFLOST=1
C
C     FILL IN FREQUENCY ARRAYS
C     BSFN(K)=NO. OF PROTOTYPES BELOW SURFACE K ACCORDING TO TEACHER
C     ASFN(K)=NO. OF PROTOTYPES ABOVE SURFACE K ACCORDING TO TEACHER
  300 IFEAT=(XX(1)*10.)+1
      JFEAT=(XX(2)*10.)+1
      IF(NWT.EQ.2) JFEAT=1
      NPRO=NPRO+1
      NPR(IFEAT,JFEAT)=NPR(IFEAT,JFEAT)+1.
      AVENPR=NPRO/RW(1)
      FAC1=AVENPR/NPR(IFEAT,JFEAT)
      IF(NT.EQ.NCLT) GO TO 315
      DO 310 K=NT,NCLM1
      BINFO(K,IFEAT,JFEAT)=BINFO(K,IFEAT,JFEAT)+1.
  310 BSFN(K)=BSFN(K)+FAC1
      IF(IFLOST.EQ.1) GO TO 402
      IF(NT.EQ.1) GO TO 325
  315 NTM1=NT-1
```

```
      DO 320 K=1,NTM1
  320 ASFN(K)=ASFN(K)+FAC1
  325 IF(NT.EQ.NS) GO TO 400
      DIV=1.
      DO 330 K=1,NWTM1
  330 DIV=DIV+XX(K)**2
      CORFAC=FAC1/SQRT(DIV)
      IF(NT-NS) 350,400,340
C
C     STUDENT HAS UNDER-CLASSIFIED
C     XCLASA(J) = PROTOTYPES BELOW SURFACE J (SAYS STUDENT)
C                                ABOVE SURFACE J (SAYS TEACHER)
  340 DO 349 J=NS,NTM1
      XCLASA(J)=XCLASA(J)+FAC1
      IF(XAMDA(J).EQ.0.) GO TO 349
      DO 348 I=1,NWTM1
      TPR(J,I)=TPR(J,I)+XX(I)*CORFAC*XAMDA(J)
  348 RPR(J,I)=RPR(J,I)+TPR(J,I)
      TPR(J,NWT)=TPR(J,NWT)+CORFAC*XAMDA(J)
      RPR(J,NWT)=RPR(J,NWT)+TPR(J,NWT)
  349 CONTINUE
      GO TO 400
C
C     STUDENT HAS OVER-CLASSIFIED
C     XCLASB(J) = PROTOTYPES ABOVE SURFACE J (SAYS STUDENT)
C                                BELOW SURFACE J (SAYS TEACHER)
  350 NSM1=NS-1
      DO 359 J=NT,NSM1
      XCLASB(J)=XCLASB(J)+FAC1
      IF(XAMDA(J).EQ.0.) GO TO 359
      DO 358 I=1,NWTM1
      TPR(J,I)=TPR(J,I)-XX(I)*CORFAC*XAMDA(J)
  358 RPR(J,I)=RPR(J,I)+TPR(J,I)
      TPR(J,NWT)=TPR(J,NWT)-CORFAC*XAMDA(J)
      RPR(J,NWT)=RPR(J,NWT)+TPR(J,NWT)
  359 CONTINUE
      GO TO 400
C
  402 NPLOST=NPLOST+1
      IFLOST=0
C
  400 CONTINUE
      IHC=MAX0(NS,IHC)
```

```
      RETURN
403   NPOOR=NPOOR+1
      RETURN
      END
```

APPENDIX B

VARIABLE SUMMARY

Variable	Description	Page
$A_{j,j+1}$	Ratio of class importances for feature vectors $\underline{F}_{j+1}$ and $\underline{F}_j$	9
B	Total number of tally bins in a problem	11
c	Pattern classification algorithm adjustment factor	13
c_i	Initial splitting surface location parameter for i'th surface	40
C_H	Highest class obtained by a branch of prototypes	12
C_N	New class	20
C_o	Old class	20
C_s	Source class	20
C_u	Source splitting class	20
D_b	Weighting factor for b'th tally bin	11
D_c	Distance to collision	19
D_s	Distance to surface intersection	19
E	Energy of a particle	5
E_{cut}	Energy cutoff	19
f_a	Normalization factor for pattern classification algorithm	15
f_b	Distribution factor for pattern classification algorithm	15
$\underline{F}$	Feature vector	7
$\underline{F}^*$	Augumented feature vector	8
$\underline{F}_j$	Feature vector of j'th prototype	9
g	Discriminant function	8
g_i	Discriminant function of i'th splitting surface	9

$\underline{H}_j$	Sum of weight vector changes after j prototypes	15
I_n	For geometry splitting, importance of region n For state space splitting, importance of class to which vector $\underline{F}_n$ belongs	4
N	Number of histories in a Monte Carlo calculation	16
N_c	Total number of classes (importance regions)	8
N_e	Splitting ratio for energy splitting	18
N_g	Splitting ratio for geometry splitting	19
N_p	Number of prototypes created from a source particle until the termination of the present track being followed	12
N_u	Number of $T_j \neq 0$ splitting surfaces	12
NS	Prototype classification according to the student	13
NT	Prototype classification according to the teacher	13
N_t	No. of new tracks created by splitting, (n,xn), or fission	25
N_w	No. of words of PROTO array previously used	25
$P(\underline{F})$	Probability density of feature vectors in feature space	16
$P(\underline{F}\|n)$	Probability that feature vector $\underline{F}$ belongs to class n	13
$P(\underline{F}\rightarrow\underline{F}' \geq n)$	Probability that feature $\underline{F}$ leads to a feature vector $\underline{F}'$ which is in class n or greater	28
P_h	No. of prototypes produced per neutron history	58
R	Dimensionality of feature space	8
R_{cost}	Relative computer cost	17
$\underline{R}_J$	Sum of all weight vectors through prototype J less $J*\underline{W}_o$	15
R_s	Ratio of source neutrons to total number of prototypes	35

$\underline{S}_J$	Sum of all weight vectors through prototype J	15
t	Time of a particle	5
t_j	Tally weight contribution of prototype j	10
t_p	Computer time/particle history	16
T_{cut}	Time cutoff	19
T_j	Total tally weight contribution from prototype j and all its progeny	12
$\hat{T}_i$	Tally weight contribution threshold for class i	12
u	Direction cosine with x-axis	5
v	Direction cosine with y-axis	5
w	Direction cosine with z-axis	5
wt	Particle weight	4
wt_j	Weight of particle used in j'th prototype	10
$\underline{W}$	Discriminant function weight vector	8
$\underline{W}_i$	Weight vector for i'th splitting surface	20
$\underline{W}_j$	Weight vector after j'th prototype	13
$\underline{W}_{i,j}$	Weight vector of i'th splitting surface after j'th prototype	10
$\overline{\underline{W}}$	Average weight vector	15
$\underline{W}_o$	Original weight vector	15
x	x-coordinate location of particle	5
$\underline{X}$	State space vector	7
y	y-coordinate of particle	5
z	z-coordinate of particle	5
σ^2	variance of a Monte Carlo estimate	16
σ_s^2	variance of the Monte Carlo sample	16
λ	learning parameter	15

APPENDIX C

FORTRAN VARIABLE SUMMARY

Variable	Description	Page
ADFWT(M,I)	The I'th component of the average weight vector $\underline{W}_J$ of the M'th splitting surface	30
BOA(I)	XCLAB(I)/XCLASA(I), the variable used to measure learning	32
BTAL	The accumulated tally weight contribution from the prototype being processed up to and including the prototype NPROTO	25
CONV	The maximum deviation of BOA from 1 required for learning	32
DFWT(M,I)	I'th component of the initial weight vector $\underline{W}_o$ of the M'th splitting surface	30
DIV	Magnitude of the vector $\underline{F}^*$	29
F(J)	J'th component of feature vector $\underline{F}$	25
ICONV	The number of prototypes that must be processed before the λ can be adjusted	31
ICONVT	The number of prototypes that must be processed before learning can be terminated.	32
IDCONV	The number of prototypes that must be processed (since the last λ adjustment) before the λ can be adjusted again	31
IHCNOD(I)	The maximum class (NS) obtained at a branching intersection.	26
JDIF	Number of prototypes being processed	25
NBNOD(I)	No. of branches remaining unprocessed on the I'th branching intersection	24
NNODS	No. of branching intersections	24
NODS(I)	The prototype No. of the I'th branching intersection	24
NPR(K,L)	Approximation used for P($\underline{F}$)	28
NPRO	Total No. of prototypes created	28

NPROTO	No. of prototypes stored in PROTO	24
NPS	No. of source particles started	32
NPSLRN	Threshold histories for using splitting surfaces	32
NS	Student prototype classification	13
NT	Teacher prototype classification	13
PROTO(I)	Array for prototype storage	24
RPR(M,I)	The I'th component for the $\underline{R}_J$ of the M'th splitting surface	29
TPR(M,I)	The I'th component for the $\underline{H}_J$ of the M'th splitting surface	29
TTAL	Total tally weight contribution for a prototype	25
TWT(I)	I'th tally bin weight	24
XAMDA(I)	Learning parameter for I'th surface	31
XCLASA(I)	Number of misclassified prototypes above surface I (according to NT)	31
XCLASB(I)	Number of misclassified prototypes below surface I (according to NT)	31
XFAC(J)	Constants used to vary λ	31
XMCLR(I)	Misclassification rate of I'th splitting surface	31

Printed in the United States of America. Available from
National Technical Information Service
US Department of Commerce
5285 Port Royal Road
Springfield, VA 22161

Microfiche $3.00

001-025	4.00	126-150	7.25	251-275	10.75	376-400	13.00	501-525	15.25
026-050	4.50	151-175	8.00	276-300	11.00	401-425	13.25	526-550	15.50
051-075	5.25	176-200	9.00	301-325	11.75	426-450	14.00	551-575	16.25
076-100	6.00	201-225	9.25	326-350	12.00	451-475	14.50	576-600	16.50
101-125	6.50	226-250	9.50	351-375	12.50	476-500	15.00	601-up	

Note: Add $2.50 for each additional 100-page increment from 601 pages up.