COMPUTER SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES
DEPARTMENT OF ELECTRICAL ENGINEERING
STANFORD UNIVERSITY · STANFORD, CA 94305

SEL 78-029

# OPTIMAL PROGRAM CONTROL STRUCTURES BASED ON THE CONCEPT OF DECISION ENTROPY

by
Ruby Bei-Loh Lee

July 1978

Technical Report 156

The work described herein was supported in part by the Ballistic Missile Defense Systems Command under contract no. DASG60-77-C-0073

SEL 78-029

# OPTIMAL PROGRAM CONTROL STRUCTURES BASED ON THE CONCEPT OF DECISION ENTROPY

by

Ruby Bei-Loh Lee

July 1978

Technical Report 156

Digital Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, CA 94305

The work described herein was supported in part by the Ballistic Missile Defense Systems Command under contract no. DASG60-77-C-0073.

SEL 78-029

Digital Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, CA 94305

Technical Report 156

July 1978

# OPTIMAL PROGRAM CONTROL STRUCTURES BASED ON THE CONCEPT OF DECISION ENTROPY

by

Ruby Bei-Loh Lee

## ABSTRACT

The ability to make decisions dynamically during program execution is a very powerful and valuable tool. Unfortunately, it also causes severe performance degradations in high-speed computer organizations which use parallel, pipelined or lookahead techniques to speed up program execution. An optimal control structure is one where the average number of decisions to be made during program execution is minimal among all control structures for the program. Since decisions are usually represented by conditional branch instructions, finding an optimal control structure is equivalent to minimizing the expected number of conditional branch instructions to be encountered per program execution,

By decision entropy, we mean a quantitative characterization of the uncertainty in the instruction stream due to dynamic decisions imbedded in the program. We define this concept of decision entropy in the Shannon information-theoretic sense. We show that a program's intrinsic decision entropy is an absolute lower bound on the expected number of decisions, or conditional branch instructions, per program execution We show that this lower bound is achieved iff each decision has maximum uncertainty. We also indicate how optimal control structures may be constructed.

The work described herein was supported in part by the Ballistic Missile Defense Command under contract no, DASG60-77-0073.

TABLE OF CONTENTS

## 1. INTRODUCTION

A program is written to solve a set of related problems. These problems are said to be "related" because they require the solutions of common sub-problems. These problems are not identical because each may involve a different subset of solutions to common subproblems. Hence, a program consists of three main entities:

(i) a collection of solutions of different subproblems (each of which may be defined hierarchically as a program);

(ii) interfaces for passing information between subproblems;

(iii) a control structure which specifies the subset of subproblems to be solved, and directs the order in which they are to be solved for each problem in the set.

Entity (i) is represented in the program as subroutines, loops, expression evaluations. Entity (ii) is represented in parameter passing and "common" variables. Entity (iii) is represented by branch instructions which alter the sequential flow of instruction execution.

The purpose of this paper is to optimize entity (iii), the control structure of a program, with respect to time. Optimality of a program, for a given set of problems. must be defined with respect to space or to time. By space-optimal, we mean that the static program size is smallest amongst all programs fcr that set of problems, By time-optimal, we mean that the dynamic program size, or the execution time of the program in numbers of instructions, is smallest amongst all programs for that set of problems Due to the well-known "space-time tradeoff", a program is seldom both space-optimal and time-optimal Although space-optimality may be crucial in applications with very limited memory, time-optimality is usually more important in for example, real-time applications and programs which are repeatedly executed. In order to achieve a time-optimal program, we need to find a time-optimal control structure for the given set of problems

Definition 1:

A time-optimal control structure is one where the average number of decisions to be made during program execution is minimal among all control structures for the program.

Since decisions are represented by conditional branch instructions in the program, finding an optimal control structure, if one exists, is equivalent to minimizing the expected number of conditional branch instructions to be encountered per program execution.

The main thesis of this paper is to show an absolute lower bound on the expected number of conditional branch instructions per program execution. This absolute lower bound is characterized as the 'decision entropy" of the program, or the uncertainty in the flow of program execution due to the dynamic results of decisions embedded in the program. This 'decision entropy" is dependent only on the characteristics of the set of problems to be solved in the program, and hence, may be calculated a priori. In general, this absolute lower bound is not attainable, and optimal control structures are those that come closest to the lower bound. We also show the special (necessary and sufficient) conditions, under which the absolute lower bound is attained.

The importance of optimizing the control structure of a program cannot be overemphasized [6-10]. For example, conditional branch instructions cause severe time degradations in computer organizations which use a pipelined instruction unit to speed up the execution of the program [6,10]. Such organizations are often referred to as "overlapped SISD (Single Instruction Single Data) organizations" [6]. The uncertain result of a conditional branch result either causes a holdup of the flow of instructions into the "execution pipe", or a flush of the "execution pipe" in the event of a wrong guess for systems which guess this runtime result.

Another example occurs in anticipatory memory management, where a look-ahead mechanism prefetches the instructions which are to be executed next, in order to minimize delays due to memory faults. Conditional branch instructions introduce uncertainty into the instruction stream which cannot be resolved a priori. Again severe performance degradations result.

## 2. PROBLEM DEFINITION AND TERMINOLOGY

Problem Definition: To find the minimum number of conditional branch instructions occurring during program execution, on the average, for any given program.

Terminology

Let the successor set of an instruction be the set of different instructions to which control may pass during program execution. Then a conditional branch instruction is an instruction where the size of the successor set is greater than one. (Assuming deterministic programs, the flow of execution passes to at most one instruction in the successor set of a conditional branch instruction.) Hence, conditional branch instructions represent decisions embedded in the program since alternate paths of action are possible during program execution. We note that an unconditional branch instruction does not represent a decision since it has a successor set size of one. Although our definition allows the size of the successor set of a conditional branch instruction to be any integer greater than one, most computers today restrict this size to two, corresponding to the binary truth values of False and True, or 0 and 1. For convenience, we will assume binary truth values for the decisions, in the rest of this paper.

For the purposes of this paper, we consider programs which are loop-free. Then the control structure of a program can be represented as a binary tree

[1], called the binary decision tree, of the program. Each internal node represents a decision, or conditional branch instruction, in the program. Each external node, or leaf, represents the end of an execution of the program. Hence, each execution traces a path, called a decision path, through this tree. Equivalently, we can say that each execution traces a control sequence through the control structure of the program. Each decision path, or control sequence represents the solution of one problem in the set of related problems the program was written to solve.

We can now distinguish between the static program and the dynamic program. By the static program, we mean the invariant body of instructions that comprise the program. By the dynamic program, we mean the set of distinct decision paths through the static program, each execution being an instance of the dynamic program. Clearly, the static program is a constant but the dynamic program is a random variable. Indeed, it is this uncertainty in the (dynamic) program as a random variable that makes it interesting to apply the concept of decision entropy, which we will define next.

Definition 2: Decision Entropy

Suppose the (dynamic) program, Y, has n decision paths, $y_1, y_2, \ldots, y_n$, each with probability $p_1, p_2, \ldots, p_n$. This is represented by the following standard notation:

$$\begin{pmatrix} y_1 & y_2 & \cdots & y_n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix}$$

Then the decision entropy of Y is defined to be:

$$H_Y = - \sum_{1 \le i \le n} p_i \log p_i$$

Clearly, this is just the Shannon definition of entropy in information theory [2]. We note that the logarithm function may be taken to any base, since changing this base amounts only to multiplication by a known constant. For the purposes of this paper, logarithms are always taken to base 2.

Suppose, now that the decision path $y_i$ involves the execution of mi conditional branch instructions, then the number of conditional branch instructions which may occur during any execution of the program Y is also a random variable, represented as follows:

$$M_Y = \begin{pmatrix} m_1 & m_2 & \cdots & m_n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix}$$

Therefore, the average number of conditional branch instructions occurring per program execution is

$$\overline{M}_Y = \sum_{1 \leq i \leq n} p_i m_i$$

We wish to show that this average number of conditional branch instruction is bounded below by the intrinsic decision entropy in the program, i.e., $\overline{M}_Y \geq H_Y$.

## 3. An Intuitive Introduction via 'Trivial' Programs

We would like to give the reader an intuitive feel for the relationship between the number of conditional branch instructions in the program and its decision entropy, for 'trivial' programs.

Consider the class of programs X with no conditional branch instructions, i.e., $\overline{M}_X = 0$. Clearly, there is only one decision path through X, namely that starting with the first instruction, proceeding sequentially to the last instruction. Hence the dynamic program is identical to the static program, both being the same constant. Since the probability associated with the single decision path is one, we have

$$H_X = 1.\log_2 1 = 0$$

Hence $\overline{M}_X = H_X = 0$.

Conversely, suppose it is known that a program has zero decision entropy. Is it then true that this program has no conditional branch instructions? The answer is no, since a program with zero decision entropy may have arbitrarily many conditional branch instructions, albeit they are all redundant. This is stated in the following summarizing observation:

<u>Observation:</u> If a program has no conditional branch instructions, then its decision entropy is zero. Conversely, if a program has zero decision entropy, then either

(i) it has no conditional branch instructions, or

(ii) it has an arbitrary number of conditional branch instructions, each of which has a successor set where one member has probability one of being the next instruction executed (i.e., all conditional branch instructions are redundant since they are essentially unconditional branches). Hence, for 'trivial' programs, it is true that $\overline{M}_\gamma \geq H_\gamma$.

We note that these classes of programs are 'trivial' only in that they have trivial decision trees. The decision tree is either null (no conditional branch instructions) or can be reducible to a null decision tree (all conditional branch instructions are redundant). Otherwise, these programs may represent very complicated and nontrivial computations.

## 4. The Main Results for All Proarams

We will now like to show the general result that $\overline{M}_\gamma \geq H_\gamma$, in theorem 1. The proof of this theorem uses the following two lemmas. In lemma 1, $\{p_i, 1 \leq i \leq n\}$ and $\{q_i, 1 \leq i \leq n\}$ represent two arbitrary probability distributions.

Lemma 1: If $0 \leq p_i, q_i \leq 1$, for $1 \leq i \leq n$, and

$$\sum_{1 \leq i \leq n} p_i = \sum_{1 \leq i \leq n} q_i = 1, \text{ then}$$

$$-\sum_{1 \leq i \leq n} p_i \log p_i \leq -\sum_{1 \leq i \leq n} p_i \log q_i,$$

with equality iff $p_i = q_i$ for $1 \leq i \leq n$.

Proof: The crux of the proof lies in observing the fact (see Figure 1), that a convex function, like logarithm, always lies below its tangent. Taking the tangent at x = 1, we have:

$\log x \leq x - 1$, with equality iff x = 1

• [illegible] $\frac{q_i}{p_i} \leq \frac{q_i}{p_i} - 1$, with equality iff qi = pi

$\therefore \sum_{1 \leq i \leq n} p_i \log \frac{q_i}{p_i} \leq \sum_{1 \leq i \leq n} (q_i - p_i)$, with equality

iff $q_i = p_i$, $1 \leq i \leq n$

● [illegible] $\sum$ [illegible] $p_i \log q_i - \sum_{1 \leq i \leq n} p_i \log p_i \leq 0$, with equality

iff $q_i = p_i$, $1 \leq i \leq n$. $\square$

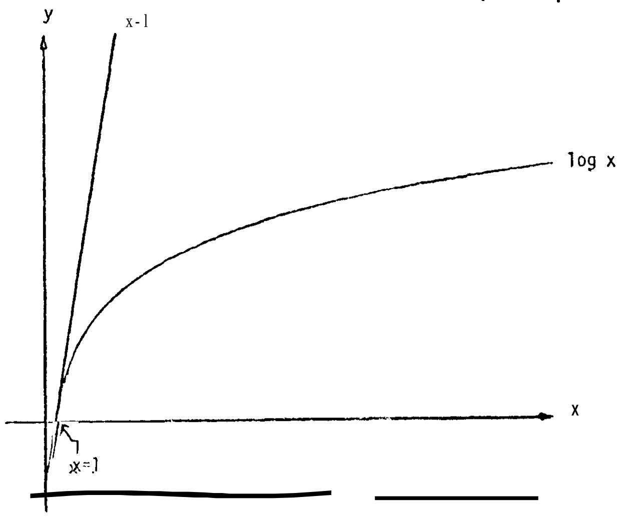


Figure 1. Relation between a convex function and its tangent at x = 1.

Lemma2: $\sum_{1 \le i \le n} 2^{-m_i} = 1$, where mi is the number of conditional branch instructions in the ith. decision path.

Proof: Let $l_j$ be the number of mi such that mi = j (i.e., $l_j$ is the number of terminal nodes on level (j + 1) in the binary decision tree, where the root is level 1 of the tree).

Let d be the maximum number of levels in the binary decision tree with at least one nonterminal node, so that $\sum_{1 \le j \le d} l_j = n$.

Then $\sum_{1 \le i \le n} 2^{-m_i} = \sum_{1 \le j \le d} l_j 2^{-j}$ (see Figure 2)

The proof is by induction on d:

When d = 1, $l_1 = 2$ and $\sum_{1 \le i \le 1} 2.2^{-i} = 1$.

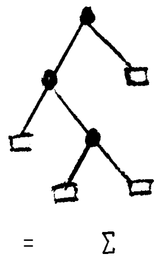
Assume true for d = k, i.e., $\sum_{1 \le j \le k} l_j 2^{-j} = 1$.

(Every time we add a new level, we add $l_{k+1}$ paths with k+1 nonterminal nodes, and delete $\frac{l_{k+1}}{2}$ paths with k nonterminal nodes.)

To show true for d = k+1 if true for d = k:

$$\sum_{1 \le j \le k+1} l_j 2^{-j} = \sum_{1 \le j \le k} l_j 2^{-j} + l_{k+1} 2^{-(k+1)} - \frac{l_{k+1}}{2} 2^{-k}$$

$$= 1 + l_{k+1} (2^{-(k+1)} - 2^{-(k+1)}) = 1. \qquad \square$$



| Level = j | $l_j$ |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 0 |

d = 3

$$\therefore \sum_{1 \le i \le 4} 2^{-m_i} = \sum_{1 \le j \le 3} l_j 2^{-j} = 1(\tfrac{1}{2}) + 1(\tfrac{1}{4}) = 2(\tfrac{1}{8}) = 1$$

Figure 2

Theorem 1: $\bar{M}_Y \geq H_Y$ with equality iff $p_i = 2^{-m_i}$, $1 \leq i \leq n$. In words, the average number of conditional branch instructions occurring per program execution is greater than or equal to the decision entropy, $H_Y$, of the program Y. This absolute lower bound is achieved iff $p_i = 2^{-m_i}$ for all i from 1 to n.

Proof: From lemma 1, we have

$$-\sum_{1 \leq i \leq n} p_i \log p_i \leq -\sum_{1 \leq i \leq n} p_i \log q_i, \text{ with equality iff}$$

$$p_i = q_i, \ 1 \leq i \leq n.$$

Before we can make the substitution $q_i = 2^{-m_i}$, we need to check that the 2 conditions on the $q_i$, as stated in lemma 1 are satisfied:

Condition (1): $0 \leq q_i \leq 1$, $1 \leq i \leq n$ is satisfied as

$$0 \leq (\tfrac{1}{2})^{m_i} \leq 1 \text{ since } m_i \geq 0$$

Condition (2): $\sum_{1 \leq i \leq n} q_i = 1$ is satisfied since

$$\sum_{1 \leq i \leq n} 2^{-m_i} = 1 \text{ from lemma 2.}$$

$$\therefore -\sum_{1 \leq i \leq n} p_i \log p_i \leq -\sum_{1 \leq i \leq n} p_i \log 2^{-m_i}, \text{ with equality iff } p_i = 2^{-m_i}, \ 1 \leq i \leq n$$

$$\therefore \sum_{1 \leq i \leq n} m_i p_i \geq H_Y, \text{ with equality iff } p_i = 2^{-m_i}, \ 1 \leq i \leq n. \qquad \square$$

Hence we have found a lower bound on the expected number of conditional branch instructions for any given program. The necessary and sufficient conditions for this lower bound to be achieved are satisfied when the result of every conditional branch instruction is 'True' with probability $\frac{1}{2}$, and 'False' with probability $\frac{1}{2}$. This corresponds to the special case of maximum uncertainty for each individual decision, which we will refer to as <u>maximum branch-result entropy</u>. Here, $H = -\frac{1}{2} \log \frac{1}{2} - \frac{1}{2} \log \frac{1}{2} = 1 = \log 2$ for each branch-result.

It is well-known that $H \leq \log n$, where $n$ is the number of outcomes of the random variable concerned. When $H = \log n$, the random variable is said to have maximum entropy. The maximum entropic case occurs iff $p_1 = p_2 \ldots = p_n$ (see [2]-[5]). Hence, the minimum expected number of decisions is achieved if each decision has maximum branch-result entropy.

A control structure which achieves the lower bound in Theorem 1 is said to be absolutely optimal. In general, we usually can not construct an absolutely optimal control structure given the probabilities $p_1, p_2, \ldots, p_n$, since if we choose $m_i$ to satisfy $p_i = 2^{-m_i}$, for all i, it may turn out that $m_i = -\log p_i$ is not an integer. However, we can certainly choose $m_i$ to be the smallest integer greater than or equal to $-\log p_i$, i.e., the ceiling function. This gives us a one-unit range for the expected number of conditional branch instructions that is actually achievable by an optimal control structure.

Theorem 2: If $H_Y$ is the decision entropy of program Y, then there exists a control structure for Y where the expected number of conditional branch instructions, $\overline{M}_Y$, satisfies $H_Y \leq \overline{M}_Y < H_Y + 1$.

Proof: Let $m_i = \lceil -\log p_i \rceil$, $1 \leq i \leq n$

(a) $\therefore -\log p_i \leq m_i < -\log p_i + 1, \quad 1 \leq i \leq n$

We need to check that the $\{m_i, 1 \leq i \leq n\}$ will correspond to a legitimate binary tree. The criterion required is that $\sum_i 2^{-m_i} \leq 1$.

Since $-\log p_i \leq m_i$, we have $2^{-m_i} \leq p_i$

$$\therefore \sum_i 2^{-m_i} \leq \sum_i p_i = 1$$

Multiplying (a) by $p_i$ and summing over i gives:

$$\sum_i -p_i \log p_i \leq \sum p_i m_i < \sum -p_i \log p_i + \sum_i p_i$$

(i.e.), $H_Y \leq \overline{M}_Y < H_Y + 1$

The observant reader will already have noted an analogy between the time-optimal control structure problem and the noiseless coding problem in information theory. This analogy is given explicitly in Appendix I. In the case of the time-optimal control structure we wish to minimize the average number of conditional branch instructions per program execution. In the case of the noiseless coding problem, we wish to minimize the average code-word length. In both cases, the average that we wish to minimize is bounded below by the uncertainty, or entropy, involved in the situation.

In the case of the time-optimal control structure, we wish to minimize the average number of conditional branch instructions per program execution in order to minimize the "perturbations" [6] in the flow of the instruction stream. As discussed earlier, these perturbations are known to cause severe time-penalties in high-speed computer organizations.

In Theorem 1 we have shown an absolute lower bound, which is none other than the decision entropy of the set of problems solved by the program. We stress that this decision entropy is independent of the machine (hardware) implementation and of the program (software) implementation. It is dependent only on the relative frequency of each problem in the given set of related problems to be solved by the program, and hence can be calculated a priori.

The existence of an absolute lower bound on the average number of decisions per program execution gives us a standard by which to compare the "goodness" of any arbitrary control structure for the given set of problems. We have shown the necessary and sufficient conditions under which this lower bound is attained, but in general, these conditions do not hold. In Theorem 2, we showed that an optimal control structure can come within 1 unit of this absolute lower bound. How can we construct an optimal control structure for any arbitrary set of problems?

## 6. CONSTRUCTION OF OPTIMAL CONTROL STRUCTURES

We wish to construct an optimal control structure for program Y which has n decision-paths, each with probabilities $p_1, p_2, \ldots p_n$, respectively, of occurring during program execution, $\sum_{1 \le i \le n} p_i = 1$. (In other words, program Y consists of a set of n related problems, each with frequency of occurrence proportional to $p_1, p_2, \ldots p_n$.)

It is clear from the analogy of the time-optimal control structure problem to the noiseless coding problem, that the construction of optimal control structures is analogous to the construction of optimal codes, i.e., codes where the expected code-word length is minimal among all codes for the given set of symbols $\{y_1, y_2, \ldots y_n\}$. Algorithm H, below, for the construction of optimal control structures (or optimal binary decision trees) is entirely analogous to Huffman's procedure for the construction of optimal (instantaneous) codes [2-5].

We assume at the beginning of algorithm H, that we are given a set Y of n elements, $Y = \{y_1, y_2, \ldots y_n\}$, each element representing a decision-path $y_i$ with associated probability $p_i$. Algorithm H will then construct an optimal binary decision tree "bottom. up", i.e., from the leaves to the root'.

Algorithm H (Construction of Optimal Control Structure)

```
While $|Y| \geq 2$ do
begin
    (1) Find the 2 elements $y_i, y_j$ in Y with the two smallest
        associated probabilities $p_i, p_j$.
    (2) Let them be the left and right sons of a common father,
        $y_{ij}$, whose associated probability is pi $+ p_j$.
    (3) Delete $y_i$ and $y_j$ from Y and insert $y_{ij}$ into Y.
end.
```

We note that Algorithm H produces a binary decision tree where the least probable decision paths have the longest pathlengths to the root. Since there are many proofs of the optimality of Huffman's procedure [2-5], we state the following theorem without proof:

Theorem 3: Given the probability distribution $\{p_1, p_2, \ldots, p_n\}$ of the n decision paths of program Y, algorithm H produces an optimal control structure for Y.

We note that the optimal control structure constructed by Algorithm H is not unique since

- In step (1), the 2 elements $y_i$, $y_j$ need not have probabilities smaller than the probabilities of all the remaining elements- When elements have equal probabilities, the choice is arbitrary among these elements.
- In step (2), the left and right sons may be interchanged without affecting the resultant optimality of the control structure.

## 7. CONCLUSIONS

The existence of dynamic decision-making facilities in the form of conditional branch instructions in a program is an extremely powerful and valuable tool for problem-solving. But if we try to speed up the execution time of a program by overlapped, lookahead or parallel execution techniques, then the potential performance is severely degraded by the uncertainties caused by these conditional branch instructions

In some SIMD-machines [6], conditional branch instructions appear to have been removed, but actually they have merely been camouflaged by masking instructions - the nature of the set of problems to be solved often requires some amount of dynamic decision-making, which we defined as the decision entropy inherent in the program.

In this paper, we showed the existence of an absolute lower bound for the average number of conditional branch instructions, or decisions, for a given program. This absolute lower bound is none other than the program's intrinsic decision entropy, as shown in Theorem 1. This absolute lower bound is attained iff each decision has maximum branch-result entropy.

In general, the absolute lower bound is not attainable and an optimal control structure for a program is one that comes closest to this lower bound. In theory, it is always possible to come within one decision of this lower bound, as shown in Theorem 2.

We also showed how to construct an optimal control structure for a program using a method equivalent to the Huffman construction of optimal codes.

## 8. REFERENCES

(1) Knuth, D. E., "The Art of Computer Programming", Vol. 1, Chapt. 2, Addison Wesley, 1969.

(2) Shannon, C. E. and Weaver W., "The Mathematical Theory of Communication", University of Illinois Press, 1949.

(3) Slepian, D., editor, 'Key Papaers in the Development of Information Theory", IEEE Press, 1973.

(4) Ash, R., "Information Theory", Interscience Publishers, John Wiley & Sons, 1965.

(5) Khinchin, A., "Mathematical Foundations of Information Theory", Dover Publications, 1957.

(6) Flynn, M. J., "Some Computer Organizations and Their Effectiveness', IEEE Transactions on Computers, September 1972.

(7) Flynn, M. J., "Trends and Problems in Computer Organizations", IFIPS Congress '74, (invited) Stockholm, Sweden, August 1974, North Holland Pub. 1975, pp. 2-10.

(8) Thornton, J. E., 'Design of a Computer System: The Control Data 6600", Scott, Foresman, & Co., 1970.

(9) Keller, R., "Look-Ahead Processors", Computing Surveys, Vol. 7, No. 4, December 1975.

(10) Ibbett, R. N., "The MU5 Instruction Pipeline", Computer Journal, 15, 1, February 1972.

## APPENDIX I

### ANALOGY TO NOISELESS CODING PROBLEM

It is interesting to show that there is a direct analogy between our Optimal Control Structure problem and the Noiseless Coding problem in coding theory (see [2]-[5]). The Noiseless Coding problem [2] consists of:

(1) A random variable $X = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix}$, where independent instances of X generate a sequence of symbols from the set $\{x_1, x_2, \ldots, x_n\}$.

(2) A set of code characters (a,, $a_2$, . . . . $a_D\}$ called the alphabet, where each symbol $x_i$ is assigned a finite sequence of code characters called the code-word for $x_i$. The collection of all code words forms a code, where code words are assumed to be distinct.

(3) The goal of the noiseless coding problem is to minimize the average code-word length, i.e., minimize $\sum_{1 \leq i \leq n} p_i m_i$, where mi is the length of the code word for $x_i$.

In the Optimal Control Structure problem, the random variable X is the dynamic program and each $x_i$ corresponds to a decision path in the program. Independent instances of X correspond to independent executions of the program.

The "alphabet" is the set of truth-values in the system. For example, in a binary system, we have D = 2, and the alphabet = {0, 1). The "code-word" for each decision path $x_i$ is the unique pathname of $x_i$ in the decision tree, where each path from one node to a successor node is labelled either 0 or 1. The "code" corresponds to the set of all pathnames of terminal nodes in the decision tree.

The goal of the time-optimal control structure problem is to minimize the average "code-word length", or the average number of decisions encountered during program execution.