C.1

AN INFORMATION THEORETIC MEASURE OF ALGORITHMIC COMPLEXITY

by

Lois Wright

Hon. B.Sc., University of Western Ontario, 1972

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the Department

of

Computer Science

We accept this thesis as conforming to the

required standard

THE UNIVERSITY OF BRITISH COLUMBIA

April 1974

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver 8, Canada

Date 24 April 1974

ABSTRACT

This work is a study of an information theoretic model which is used to develop a complexity measure of an algorithm. The measure is defined to reflect the computational cost and structure of the given algorithm. In this study computational costs are expressed as the execution times of the algorithm, where the algorithm is coded as a program in a machine independent language, and analysed in terms of its representation as a pseudograph. It is shown that this measure aids in deciding which sections of the algorithm should be optimized, segmented or expressed as subprograms. The model proposed is designed to yield a measure which reflects both the program flow and computational cost. Such a measure allows an 'optimal' algorithm to be selected from a set of algorithms, all of which solve the given problem. This selection is made with a more meaningful criterion for decision than simply execution cost. The measure can also be used to further analyse a given algorithm and point to where code optimization techniques should be applied. However it does not yield a method of generating equivalent algorithms.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEGEMENTS

I wish to express my sincere appreciation to Dr. Abbe Mowshowitz for his continued guidance and invaluable advice.

I would also like to thank Dr. D. Seeley for his several suggestions.

Financial assistance was received from both the National Research Council and the Department of Computer Science.

CHAPTER I: RECENT STUDIES IN ALGORITHMIC COMPLEXITY

1.1 Introduction

The following is a study of an information theoretic model which is used to develop a complexity measure of an algorithm. The measure is defined to reflect the computational cost and structure of the given algorithm. In this study computational costs are expressed as the execution times of the algorithm, where the algorithm is coded as a program in a machine independent language. For purposes of analysis, the algorithm is represented by a pseudograph. It will be shown that this measure of complexity aids in deciding which sections of the algorithm should be optimized, segmented or expressed as subprograms. The model proposed is designed to yield a measure which reflects both the program flow and computational cost. Such a measure allows an 'optimal' algorithm to be selected from a set of algorithms, all of which solve the given problem. This selection is made with a more meaningful criterion for decision than simply execution cost. The measure can also be used to further analyse a given algorithm and point to where code optimization techniques should be applied. However it does not yield a method of generating equivalent algorithms.

Before a definition of the model is given, a brief summary will be presented of the problem under study and of related research in this area. The main objectives in the study of algorithms have been to generate 'equivalent' algorithms, to

optimize compiler code through an analysis of the algorithm and, through a measure of efficiency, to compare, rank and then select, an algorithm which is in some sense optimal for a given task. Studies in the first area have often been quite theoretical and have avoided the problem of obtaining a measure of optimality. The second area, being concerned with compilers, has tended to concentrate on improvements in particular types of code rather than evaluations of the program as a whole. Investigations of the third type have often focused on only a single facet of complexity rather than the overall complexity of the algorithm. The discussion of some of the relevant studies is given in three parts. First those which have not included graphs in their analyses, then those that have used graphs, and finally those that have defined an information measure on graphs and/or algorithms are presented.

1.2 Related Work

1.2.1 Studies Without The Use Of Graphs

An early, quite theoretical evaluation of algorithms is given in a study by Ianov [17,18]. The notion of 'program schemata' is introduced to represent abstract algorithms or programs, where programs are represented in a linear notation and also as matrices. Using program schemata, a formalism which allows for transforming these schemata into equivalent ones, a decision procedure for determining equivalence, and a method of generating all equivalent schemata are developed. Rutledge [28] simplifies Ianov's [17,18] formalism by interpreting it in such

a way that the equivalence question is seen to be the equivalence problem of finite automata, for which a solution exists, even though it is rather impractical. Furthermore, a method for generating all program schemata equivalent to a given program schemata is presented. These studies are more concerned with the formalisms introduced than with an evaluation of the algorithms. But since these procedures do yield all equivalent program schemata, an efficiency measure defined on them would allow for the optimum of all equivalent programs to be determined. However, such an abstract measure cannot reveal much about code implementation or program evaluation. Thus, such studies remain strictly theoretical and very little practical work has followed from them. One exception to this is Paterson's [25] work on program schemata. He discusses the possibility of applying complexity considerations in program optimization but does not develop the necessary machinery.

In a more restricted study, Beus [9] evaluates sort algorithms by defining an efficiency measure on the number of comparisons made. However, this measure lacks sufficient generality by neglecting other cost inducing factors such as memory accesses, index calculations etc.

At a less theoretical level, Nievergelt [24] considers the problem of optimizing a program. As motiviation for his study he discusses the application of optimizing in areas where syntactic improvements would be useful, leaving semantic considerations to the programmer. His study gives several

specific practical transformations that can be applied to any program. These yield definite improvements in execution cost units. Yet the advantage gained is questionable considering the expense involved in applying such improvements to sections of the program that are seldom executed. An improvement on Nievergelt's [24] analysis appears in a much referenced paper by Allen [4], where topological properties of the program are used to obtain program modules. The most frequently executed of these are considered for optimization. Hopkins [16], using a similar approach, presents a series of transformations for optimizing programs, and the overall implementation of such techniques.

Many other papers [7,15,22] written during the 1960's and early 1970's study the problem of optimization; however, their approach is even more specialized than those mentioned above. These studies give techniques for optimizing general code, but fail to consider program flow, algorithmic implementation etc. This approach does not attempt to evaluate the program as a whole, but is concerned with improving particular sections of code. In the following section, we consider papers which take a global view of programs and algorithms.

1.2.2 Graph Theoretic Approach

One of the first graph theoretic approaches to the evaluation of algorithms is the one developed by Karp [19] in 1960. Karp notes that previously the seemingly natural

relationship between program flow and graphs had been ignored. In this paper an algorithm is expressed as a flow chart, an execution of the algorithm defines a path through the flow chart and this path is considered as a graph consisting of operational elements (seguences of instructions without a transfer) and decision elements. The graphic representation facilitates the identification of some simple program improvements. The programs are analysed in terms of subprograms, and a Markov model is introduced for investigating the frequency of execution of parts of the program. Schurmann [29] uses the graphic representation to study the problem of program segmentation. This is also an optimization problem since in order to insure fast execution, a minimum amount of paging is required. The loops in the graphic representation of the algorithm are analysed using the adjacency matrix of a partial graph based on the cycles (loops of the algorithm). The cut having the minimum number of program loops spanning it is found and this defines the optimal cut. Berztiss [8] improves on the method of construction of this matrix.

In a more recent paper, Aho and Ullman [1] introduce cost considerations into algorithmic optimzation. An algorithm is represented as directed acyclic graphs, and four transformations applicable to this representation are defined. Properties of the programs equivalent under the transformations are discussed, as well as possible extensions of this representation to further program optimizations.

Bachmann [6] recently has assessed the problem of program efficiency using an abstract calculus based on a directed graph representation of the program. Several transformational rules are given together with a measure of efficiency defined on the probabilities and the cost of all operations executed on a given path. However this is assumed to be a theoretical criterion only, since determination of probabilities is difficult and the number of distinct paths often quite large. As well, the formalism needed to apply the transformations is not mentioned, but would presumably be quite cumbersome.

Allen [3], using a graphic representation of an algorithm, proposes a useful method of analysing its control flow. In his paper, Allen defines dominance relations among nodes and from these obtains intervals (subgraphs with single entry point h such that each closed path passes through h) which are used to partition the graph and to give a partial ordering of the intervals. Mention is made of how these constructs can be used in analyses involving searches for redundant instructions, variable definitions and use relationships etc. Applying the flow anlysis methods given by Allen [3], Cocke [10] defines a set of Boolean variables on computations. This set yields a system of equations, which, when solved, point to those expressions which should be eliminated. Also included in this paper is the notion of node splitting when a cycle is entered from more than one node, a procedure often very useful in the optimization of cycles.

1.2.3 Information Theoretic Approach

Information theory, although applied to diverse problems, has not been used widely in the area of algorithmic complexity and optimization. In a 1973 paper, Green [13] defines an entropy function to measure the complexity of paths in rooted trees. Let T_n be a tree with n terminal nodes $v_1, v_2, \ldots, v_n$ and m non-terminal nodes $u_1, u_2, \ldots, u_m$, and

$$P_i = P(v_o, v_i) = P(v_o, u_{i_1}, u_{i_2}, \ldots, u_{i_k}, v_i),$$

a path of nodes from v_o to v_i. Let od(u) denote the outdegree of node u. Then the path entropy E is given by

$$E(P_i) = \sum_{j=1}^{K} \log od(u_{i_j}) + \log od(v_o);$$

and the entropy of the tree is defined by

$$E(T_n) = \sum_{i=1}^{n} E(P_i).$$

Finally, the normalized entropy of T_n is given by

$$H(T_n) = (1/n)\, E(T_n).$$

A 1969 paper by Warshall [31] analyses the problem of algorithmic complexity using information theory, under the constraint that the algorithms studied are viewed as arrays of choice-making elements. Thus, a sequence of states or Boolean expressions over states cannot be evaluated using this model. The finite input set consists of the set of data inputs $\{I_j\}$, each of which determines a sequence of states from entry to exit. An algorithm A is a function from the input values to the set of E-sequences (finite sequence of at least three elements,

beginning and ending with E, the entry or exit state). The states of A consist of the union of the elements occurring in the E-sequences. To each input I_j there corresponds a probability $P(I_j)$. The cost associated with state s is

$$-p(s\rightarrow t)\ \log\ p(s\rightarrow t)$$

where $p(s\rightarrow t)$ is the probability that t is the next state. The total indeterminancy of algorithm A, i.e. the choice among paths c, is given by

$$H(c) = -\Sigma\, p(c)\,(\log\ p(c)),\quad p(c) = \Sigma\, p(j).$$

Then letting $\Pi_c(s)$ be the number of occurrences of states s in c, the mean number of occurrences of s per path is

$$\Pi(s) = \Sigma\, p(c)\Pi_c(s)$$

and per execution cost of algorithm A is

$$C(A) = -\Sigma\, \Pi(s)\ \Sigma\, p(s\rightarrow t)\log\ p(s\rightarrow t).$$

Thus, H(A) is the algorithm cost assuming free bookkeeping, C(A) the cost (in the information theoretic sense) without this assumption. Based on the above, Warshall evaluates those algorithms which consist only of decision elements and develops some properties of this representation. Rather than present a general analysis of algorithms, this paper selects a subset of algorithms and fits it to an information theoretic model. To be of more general use, this model will have to be extended to include a wider class of algorithms, node execution costs, types of loops etc.

Mowshowitz [23] defines the structural information content of a graph using the orbits of its automorphism group. This measure is structural in the sense that it gives the information

content of a particular graph relative to a system of transformations for which it is invariant. Using finite undirected graphs and digraphs and a particular type of infinite graph, properties of this measure are investigated and the effect of several graph theoretic operations on the measure examined. It is suggested that an information measure on the structure of the graph of an algorithm might be useful in characterizing the relative complexity of the algorithm. Consideration of that proposal is the basis of this thesis.

In the following, an information value will be defined on an algorithm, methods of obtaining this value given and its usefulness as a measure of efficiency and complexity discussed.

In Chapter II definitions of necessary terminology, the cost probabilities and the information value will be given. A general explanation of the concepts related to this particular measure is also included.

In Chapter III, a procedure will be presented for computing the non-isomorphic, reduced sequences of the algorithm being studied. In addition, the chapter elaborates on the usefulness of this measure through extensive examples and possible extensions of the work.

CHAPTER II: AN INFORMATION THEORETIC MODEL OF ALGORITHMS

2.1 Constituents of an Information Theoretic Model

In order to define an information theoretic model, one must identify a channel with inputs and outputs, and input and channel probabilities (see, for example, Ash [5]). The basic channel model defines the input as members of some finite set $\{b_1, b_2, \ldots, b_J\}$, and the output as members of the same or a different set, say $\{a_1, a_2, \ldots, a_K\}$. Each output a_K is statistically dependent only on the corresponding input b_j. Such dependence is determined by a fixed conditional probability assignment $P(a_K | b_j)$, defined for each input b_j and output a_K. The set of these conditional probabilities defines the channel. Probabilities are also given for each input b_j. Such a model can be depicted by the flow diagram shown in Figure 1.

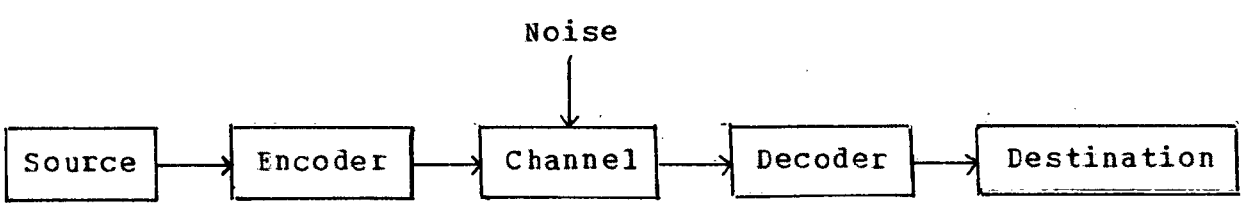


Figure 1: Information Channel

This system describes the flow of information from a source to a destination in probabilistic terms. The source message is associated with some object which can be sent over the channel. This channel is considered as the medium over which the coded

message is transmitted. The decoder operates on the channel output in an attempt to obtain the original message. In our model, the source message will be the basic steps of the algorithm; and the received message, the collection of these steps resulting from the transmission of the source message over the channel (i.e. the result of the algorithm being executed). Knowing the output, we will attempt to extract information concerning the source message which will indicate how a more efficient algorithm can be obtained.

The fundamental notion of information provided about the event x by the occurrence of the event y is defined on the model by

$$I(x;y) = \log(P(x|y)/P(x))$$

where the base of the logarithm is usually taken to be 2. The self-information of the event x is defined by

$$I(x) = -\log P(x)$$

and the conditional self-information of x given y by

$$I(x|y) = -\log P(x|y).$$

The average values of these are defined as the entropy of the ensemble X and conditional entropy of the ensemble X, given Y, respectively, and are given by

$$H(X) = -\sum P(x) \log P(x)$$

$$\text{and } H(X|Y) = -\sum P(x,y) \log P(x|y).$$

The average information between X and Y is the entropy of X less the conditional entropy of X given Y, i.e.

$$I(X;Y) = H(X) - H(X|Y).$$

Such a model will be used in the following study. The

underlying probabilities of the model derive from two different sources: data items (relative frequencies), and a structural decomposition. The latter arise in the following way. Let n_i for $1 \leq i \leq k$ be non-negative integers associated with an algorithm, and let $n = n_1 + \ldots + n_k$. A probability scheme is constructed by taking $p_i = n_i/n$. The entropy $H(p_1, \ldots, p_k) = -\sum p_i \log p_i$ of the probability scheme then serves as a measure of the structure of the algorithm. The role of the relative frequencies of data items will become clear in the detailed development which follows.

2.2 Basic Terminology

2.2.1 Graph Theory Definitions

Certain graph theoretic concepts are of use in defining this model. The following definitions are taken from Harary [14]. A graph G is a finite non-empty set V of p points together with a set X of q unordered pairs of distinct points of V. Each pair $x = \{u,v\}$ of points in X is a line of G. A graph G is directed if the set X is ordered. The elements of X are directed lines or arcs. A loop of a graph is a line which joins a point to itself; if more than one line joins two points, such lines are called multiple lines. A directed graph which allows both loops and multiple lines is called a pseudograph. A subgraph of G is a graph having all its points and lines in G. A walk of a graph G is an alternating sequence of points and lines $v_0, x_1, v_1, \ldots, v_{n-1}, x_n, v_n$, beginning and ending with points, in which each line is incident with the two points immediately

preceding and following it. Such a walk joins v_o and v_n and may be denoted by $v_o, v_1, \ldots, v_n$. The walk is <u>closed</u> if $v_o = v_n$. It is a <u>trail</u> if all the lines are distinct, and a <u>path</u> if all the points and hence, all the lines, are distinct. If a walk is closed and its $n \geq 3$ points distinct, then it is a <u>cycle</u>.

2.2.2 Model Definitions

Before defining the model precisely, we will discuss the interpretation of the term algorithm, as used in this study. An algorithm X can be represented, quite naturally, as a pseudograph G(X). Interpreted in this way, the basic elements or steps of the algorithm become the nodes, $\{v_j\}$, of a graph. Each node of the graph is called a calculation node or a decision and calculation node, depending on the type of the equivalent step in the algorithm. An arc joins two nodes in the graph if the corresponding steps are sequential in the algorithm. Each execution of the algorithm then defines a walk in G(X); these walks are, in fact, pseudographs. In any algorithm there are distinguished nodes, at which any necessary initialization of values is done. The next node, v_s, marks the beginning of the non-initialization steps of the algorithm. We give special attention to certain walks of G(X). A walk in G(X) beginning with v_s and up to but not including the next occurrence of v_s is called a <u>flow-sequence</u> a of G(X). The set $\{a_k\}$ of flow-sequences in G(X) will be denoted by A(X), or simply A. If v_d is the first decision node of a flow-sequence $a_k = v_s, v_{k_1}, \ldots v_{k_\ell}, v_d, \ldots, v_{k_p}$ then a <u>flow-subsequence</u> b of a_k is

defined as (1) any subwalk of a_K, beginning with a decision node v', and up to, but not including, the next occurrence of either v' or v_S, or (2) the subwalk $v_S, v_{K_1}, \ldots, v_{K_q}$. The introduction of flow-subsequences into the model is a structural consideration. For a given node, each of the flow-subsequences containing it provides a distinct structural context in which it may be analysed. We denote the set of all flow-subsequences of flow-sequences in G(X) by B(X) or B. The set of flow-subsequences of a flow-sequence a is denoted by B(a). In the following, we assume the total number of flow-sequences and flow-subsequences to be m and n, repectively. The notion of flow-subsequence is somewhat similar to that of interval, given by Allen [4].

Before defining reduction and isomorphism of flow-sequences, we introduce some further terminology.

In order to provide a weight function for the model, a hypothetical assembly language is defined. The language is simple so as not to incorporate any instructions dependent on the configuration of some machine. The actual instruction set is given in Appendix A. Each node v of the graph is assigned a weight, w(v), which is the cost in execution units, (i.e. machine cycles), of the assembly language instructions necessary to compute this step. The weight of a walk is the sum of the costs of its constituent nodes. For this study, we equate an increase in efficiency with a decrease in these execution costs.

Next we define the process of <u>structuring</u> to include

alterations to the program code which depend on flow relationships among certain nodes of the graph. A structuring is considered beneficial if the efficiency of the program is increased. Writing a section of the algorithm as a subroutine, or as a form that reflects a fixed order among certain nodes are examples of structuring. Structural complexity is identified with the flow of control of the algorithm. For example, we regard as simple, a structure with simple linear flow, and as complex, one with imbedded looping and branching.

We now continue with the model definitions. An implementation M(X) of algorithm X, is a coding of the algorithm as a program in the assembly language. The standard implementation, $M_0(X)$, is the initial coding of the algorithm. For $i \geq 1$, $M_i(X)$ will denote an implementation which has been structured in an attempt to decrease overall execution costs. We next make precise the terms reduction and isomorphism. A flow-sequence may contain several copies of a given flow-subsequence, as is the case, for example, when a loop is repeated several times. Noting this, we define a', the reduced flow-sequence of flow-sequence a, as the walk consisting of the distinct flow-subsequences occuring in a. We define the weight of a flow-sequence as the weight of the corresponding reduced flow-sequence. Thus, w(a) is to be interpreted throughout as w(a'). The set of all reduced flow-sequences will be denoted by A'(X) or simply A'. A flow-sequence a_r is isomorphic to flow-sequence a_s if,

a) $B(a_r') = B(a_s')$, i.e. the order of flow-subsequences within

flow-sequences is not important, or

b) $B(a_r') \subseteq B(a_s')$, and there is no a_t' such that

$B(a_r') \subseteq B(a_t')$ and

$w(a_r') \leq w(a_t') < w(a_s')$.

That is, the flow-subsequences of a_r' are also in the set of flow-subsequences of a_s', and the cost in execution units of a_r' is closer to the cost of a_s' than to any other reduced flow-sequence.

Reduction and isomorphism are present in the model in order to include structure in the measure of complexity. If the flow-sequences are such that flow-subsequences are not repeated, i.e. no reduction is possible, then within these flow-sequences, structuring of the program so that some inner loop is made very efficient is unlikely to decrease the execution times significantly. This follows since no single part of the flow-sequence is repeated often. However if much reduction is possible, i.e. inner loops are repeated many times, structuring the program to reflect this, or coding these loops more efficiently, decreases the overall execution time.

Similarly, when analysing isomorphism, if the same basic structure is repeated, i.e. one flow-sequence is simply a subset of another, then the complexity of the total graph is decreased, while the cost remains constant. Emphasis on such a flow-sequence, when attempting to optimize the code, is likely to decrease overall cost to a greater extent than a flow-sequence which has no corresponding isomorphic flow-sequences. Also, if

there are no isomorphic flow-sequences, then there is no particular flow-sequence which should be studied for more efficient coding, and the cost is unlikely to be significantly reduced by considering the structure alone. That is, assuming the unstructured program has been coded efficiently, no structuring will markedly reduce the overall cost. This will become more evident when the information value is defined below.

One can associate with an algorithm X the set of data D(X) on which the algorithm X operates. For example, if S is an algorithm to sort n numbers into order, then D(S) is the set of all n factorial possible orderings of n numbers. If D(X) is not finite, then for this analysis a finite subset D'(X) of D(X) is selected which is representative of the data on which X executes. By representative we mean that for each possible flow-sequence of G(X), there is an element of D'(X) which yields that flow-sequence. Since a finite data set can be obtained for any algorithm, we will assume that D(X) is finite in what follows. With each element d of D(X), there is an associated relative frequency, R(d), which denotes how frequently the algorithm executes on this element. For each d, let $G_d(X)$ denote the pseudograph resulting from the execution of the algorithm on datum d. We then define A_d to be the set of flow-sequences of $G_d(X)$. A_d' is the subset of A_d consisting of the reduced, non-isomorphic flow-sequences of A_d. The union of all A_d' over D(X) is the set A', defined above. B_d is the corresponding set for flow-subsequences.

2.3 Definition of the Information Model.

The model used will follow the description given in section 2.1, altered to include a probability scheme defined on the cost and structural properties of the algorithm. With the flow-subsequences as the source message and the resulting flow-sequences as the received message, this model becomes meaningful. The execution of the algorithm on D(X) produces as output, flow-sequences; an analysis of these yields information about the input which can point to a more efficient implementation of the algorithm. Thus, by transmitting the input over the channel, the algorithm can be characterized by the resulting flow-sequences, and an information value defined to reflect its cost and structure.

Given an algorithm X, its graph G(X), and its data set D(X), the model is characterized briefly as follows.

The channel is defined by the conditional probabilities induced by D(X). The input, $\{b_j\}$, is the set B of flow-subsequences of G(X). The output is given by $A' = \{a'_k\}$, the set of reduced, non-isomorphic flow-sequences obtained by transmitting B over the channel.

2.4 Definition Of Probabilities

The input, channel and output probabilities will now be defined, completing the formulation of the model. The probabilties are defined as a function of execution cost units

in order that the the model will in fact be a function of cost. Relative frequencies are assumed to be defined on D(X) and hence are incorporated in the definition of the information value. Given that the execution cost of flow-subsequence b_j is defined by

$$N_j = w(b_j),$$

and the total cost of the n flow-subsequences by

$$N = \Sigma N_j,$$

then the input probabilities are defined by

$$P(b_j) = N_j/N.$$

This gives the cost frequency of flow-subsequence b_j with respect to the total cost of the flow-subsequences. Defining

$$a^j = \{ a \mid b_j \text{ is in } B(a) \},$$

and the weight of a^j as

$$w(a^j) = \Sigma w(a),$$

where the sum is over those a in a^j, then the channel probabilities are

$$P(a_K \mid b_j) = \begin{cases} w(a_K)/w(a^j) & \text{if } b_j \text{ is in } B(a_K) \\ 0 & \text{otherwise} \end{cases}$$

i.e. the cost frequency of flow-sequence a_K with respect to those flow-sequences containing flow-subsequence b_j. We denote the output probabilities by

$$P(a_K) = \sum_{j=1}^{n} P(a_K \mid b_j) P(b_j),$$

i.e. the cost probability of flow-sequence a_K averaged cost wise over all flow-subsequences in flow-sequence a_K. For each element d of D(X), R(d) is the relative frequency of datum d.

We now show that these definitions do in fact yield probabilities.

Obviously

$$\sum_{i=1}^{n} P(b_i) = 1 \quad \text{since,}$$

$$\sum_{i=1}^{n} P(b_i) = \left[\frac{N_1}{N} + \frac{N_2}{N} + \cdots + \frac{N_n}{N}\right] = \frac{N}{N} = 1 .$$

Also

$$\sum_{k=1}^{m} P(a_k) = \sum_{k=1}^{m} \sum_{j=1}^{n} P(a_k | b_j) P(b_j)$$

$$= (1/N) \sum_{k=1}^{m} \sum_{j=1}^{n} (w(a_k)/w(a^j)) N_j$$

$$= \frac{1}{N}\left[w_{1,1} \frac{N_1}{w(a^1)} + w_{1,2} \frac{N_2}{w(a^2)} + \cdots + w_{1,n} \frac{N_n}{w(a^n)}\right.$$

$$+ w_{2,1} \frac{N_1}{w(a^1)} + \cdots + w_{2,n} \frac{N_n}{w(a^n)}$$

$$+ \cdots$$

$$\left. + w_{m,1} \frac{N_1}{w(a^1)} + \cdots + w_{m,n} \frac{N_n}{w(a^n)}\right]$$

$$= \frac{1}{N}\left[\frac{N_1}{w(a^1)}\,(w_{1,1} + w_{2,1} + \ldots + w_{m,1})\right.$$

$$\left. + \ldots + \frac{N_n}{w(a^n)}\,(w_{1,n} + \ldots + w_{m,n})\right]$$

$$= (1/N)\ (N_1 + N_2 + \ldots + N_n)$$

$$= 1$$

where $w_{K,j} = \begin{cases} w(a_K) & \text{if } b_j \text{ is in } B(a_K) \\ 0 & \text{otherwise} \end{cases}$

and $(1/w(a^i))\sum_{k=1}^{m} w_{K,i} = (w(a^i)/w(a^i)) = 1$.

In defining the model probabilities as cost measures, the concepts of cost and complexity are strengthened within the model. If a flow-sequence a_K has high probability, then this indicates one or more of the following. First, a_K consists of seldom repeated flow-subsequences, i.e. the flow-subsequences of a_K do not appear in many other flow-sequences, so structuring of the program around their occurrence in a_K will not greatly impede the execution of the other flow-sequences, and should decrease the execution time of a_K. Also, if the flow-subsequences in a_K are costly in relation to those in other flow-sequences, high probability results. This is consistent with the notion that a very costly flow-sequence should receive considerable analysis, since it necessarily will be a large contributor to overall cost (assuming uniform relative

frequencies). If the above occur, a higher cost probability will result, and hence, as will be shown, a larger information value.

2.5 Definition Of Information Measure

Using the information theoretic relation

$$I(X;Y) = H(Y) - H(Y|X),$$

the information value of an algorithmic implementation will be defined in terms of the entropy $H(A(X))$ of the ouput flow-sequences, and the conditional entropy $H(A(X)|B(X))$, of the flow-sequences given the input flow-subsequences. For each possible datum d in $D(X)$, with relative frequency $R(d)$, $H_d(A(X))$ and $H_d(A(X)|B(X))$ are calculated as

$$H_d(A(X)) = -\sum (w(a_k) P(a_k) \log P(a_k))$$

where the sum is over the set of reduced flow-sequences A'_d, and

$$H_d(A(X)|B(X)) = -\sum(N_j P(b_j) P(a_k|b_j) \log P(a_k|b_j)),$$

the sum taken over the set of flow-sequences A'_d, and flow-subsequences B_d. The costs, and hence probabilities, are those determined by the given implementation, $M(X)$. Where no confusion will arise, we will write A and B for $A(X)$ and $B(X)$, respectively.

We interpret the entropy as a measure of the cost and structural complexity of the algorithm. A lower entropy occurs if there are only a few flow-sequences in A'_d compared to the number possible, i.e. there were many isomorphic flow-sequences. Thus, proper structuring should decrease the overall cost.

Also, if the same flow-subsequence occurs in many different flow-sequences, the entropy is lower, indicating that this flow-subsequence should be coded more efficiently, with emphasis on its flow pattern. However, entropy is higher when either the flow-subsequences within certain flow-sequences are costly relative to other flow-subsequences, or flow-sequences are long and structurally complex relative to other flow-sequences. Attention should be given to such flow-sequences when attempting to optimize the code of a given algorithm, since overall cost is dominated by them.

This measure then, points to flow-sequences or flow-subsequences which should be considered for possible optimization.

The weighted conditional entropy, $H_d(A|B)$, is a refinement of the entropy measure. It considers the amount of cost and structural complexity preference that is, in a sense, duplicated. If the same flow-subsequence b appears in many different flow-sequences, then, although the code of b can be optimized, only one of the flow-sequences containing b can be structured so as to optimize on b's relative position among other flow-subsequences of the flow-sequence. This follows from the observation that if more than one of these flow-sequences has the same general structure, then they are isomorphic. But if they are isomorphic, then only one copy is present. Necessarily then, the structures in the non-isomorphic sequences are distinct, verifying the above remark. Hence, by subtracting

the conditional from the unconditional entropy, a correction is made for the amount of 'structural information' which is repeated. Also of note is the fact that if the conditional entropy measure is low, then the flow-subsequences are generally not repeated in many flow-sequences. This indicates a fairly simple structure which yields a substantial cost decrease when it is properly structured and efficiently coded. However, a high conditional entropy denotes a more complex structure, with the same flow-subsequences occurring in many structurally different flow-sequences. In a sense the conditional entropy accounts for the intersection of flow-sequences, through common flow-subsequences.

The above discussion is concerned with only a single element d of D(X). To complete the model, the input must be transmitted through the channel, i.e. the algorithm must be executed on all data. Thus the entropy and conditional entropy are defined as

$$H(A) = \sum R(d) H_d(A) \text{ and}$$

$$H(A|B) = \sum R(d) H_d(A|B),$$

where the sum is over all d in D(X). Then the information value of the implementation of algorithm X is given by

$$I(A;B) = H(A) - H(A|B).$$

2.6 Uses of The Information Value

The term cost comparable is used here to describe those algorithms which, in their standard implementations, have within

a few per cent the same execution cost per node, total number of nodes, and execution cost per datum. The maximum value of I(A;B) over a set {Z} of cost comparable algorithms which solve the same problem, is obtained from the most efficient algorithm. A higher information value on this set is more 'informative', in that it indicates any of the following. First, the program can be structured to take advantage of a dominant flow-sequence which is structurally simple. Secondly, certain flow-subsequences can be considered for optimization of code since they are dominant throughout the program. And lastly, certain flow-sequences, consisting of costly flow-subsequences, are making the program expensive to execute. In the set {Z}, an algorithm such that no structuring is beneficial to it, must consist of many distinct flow-sequences, since there are no isomorphic flow-sequences. Furthermore, no reduction is possible and some flow-subsequences appear in many different flow-sequences. Hence, both the conditional and flow-sequence probability will be low, and thus, also the information value. But an algorithm where this is not the case, i.e. many dominant flows, will have a higher information value.

On the other hand, restricting our attention to a particular algorithm X of {Z}, the best implementation M(X) is the one having the lowest information value. This follows by observing that the implementation to which consideration of structure has been most beneficial, will have the lowest cost, and hence information value. Intially the information value is calculated relative to the standard algorithm implementation

$M_o(X)$. Then, upon structuring, if the information value increases, such a structuring does not reflect the flow of the algorithm. But if the information value decreases, the given implementation included structuring and optimizing which decrease the cost significantly. A further discussion and verification of this fact is given in Chapter III, where it is shown that the information value can be used to select that implementation which is most appropriate in a given situation.

2.7 Information Value - A Complexity and Efficiency Measure

In order to analyse an algorithm, a means of ranking it relative to other algorithms for the same task must be available. This measure should include the cost of executing the algorithm, and reflect its flow pattern. Measures which are a function of a single cost contributor, although helpful when comparing algorithms relative to this factor, are necessarily, as a general measure of cost or complexity, only partially effective. Similarly, measures which consider flow only are most helpful in analysing the problem of whether and how code can be optimized, but in such study, relative costs are often disregarded, again leaving the measure, in some sense, incomplete.

A measure then, that is useful in a general sense should reflect both of these factors. In the measure given above, an attempt has been made to incorporate cost and some notion of structural complexity. An information theoretic measure seemed

to be a most natural means of combining the two entities. Through the definition of isomorphism and reduction, and of the probabilities, this model of the execution of an algorithm becomes a function of structural complexity and cost, respectively. As desired, an algorithm which is more suitable for structuring will have a higher information content than an algorithm with comparable cost, but for which it is not practical to attempt to establish an efficient structure. If algorithms are 'structurally equivalent' in the sense that the flow pattern of both indicates that the same amount of structuring is applicable, then the information value will rank the algorithms according to cost, the more expensive one having a higher information value. Thus, when comparing different implementations of some algorithm X, that one with the smallest information value will be the most efficient in terms of execution costs. Structural equivalence is indicated by the conditional entropy, which, when analysing a given algorithm, partitions the data set into blocks, where each block responds in approximately the same manner to proper structuring. If the conditional entropies are equal for elements d1 and d2 of D(X), then their walks through the algorithm are the same. This concept is useful when selecting the most costly of those sections of code that respond similarly to given code optimization. Thus, the measure achieves the goal of establishing a complexity measure which is a function of both cost and structure. This measure, by analysing the output of flow-sequences, provides information about the constituent flow-

subsequences; such information then points to ways in which the algorithm can be made more efficient. Further discussion of this observation follows in Chapter III, where several examples of how this measure has been applied are given.

CHAPTER III: AN INFORMATION THEORETIC ANALYSIS OF ALGORITHMIC COMPLEXITY

3.0 Introduction

In this Chapter, we present a more detailed study of the model. We describe the processes used to obtain its components, investigate certain properties of the defined measure, and finally, demonstrate, through examples, some of its applications.

Let X be an arbitrary algorithm. Throughout the following discussion, the nodes $v_0, v_1, \ldots, v_r$ of G(X) will be numbered to follow the directed flow, depth first. Each node is either a decision and calculation node, or simply a calculation node. Using the procedures given below, the input, output, and input and channel probabilities can be obtained, and hence, the model made operational.

3.1 Procedure for Flow-Sequence Construction

In the statement of the procedure, j is used as the index on the nodes of X. The first non-initialization step of the algorithm is v_s, and the terminal node is v_t. Here we assume that no decision node occurs in the initialization process. A stack is used to store that portion of a flow sequence that has been constructed prior to the occurrence of the present decision node. Also stored on the stack is the number of branches exiting from this decision node (and hence the number of flow-sequences that will be created).

Step 1: Initialization step.

Initialize k to 1, j to 0.

Step 2: Intitialization Nodes.

If j<s, increment j and return to step 2. If j=s, set a_k to v_s, and if v_s is a decision node, stack j and a_k; increment j, and go to step 3.

Step 3: Main Flow-Sequence Construction.

If j>r, go to step 4. Adjoin v_j to a_k. If j≠t, and v_j is a decision node, stack j, a_k; if j=t, and the stack is not empty, increment k, remove a_k, j from the stack; increment j; go to step 3.

Step 4: Terminate.

3.2 Flow-Subsequences; Weight Assignment.

The set B of flow-subsequences is formed from A in the following manner, where initially the set B is empty. For each flow-sequence a, of A, add to B any of a's flow-subsequences not already in B. To allow the analysis programs to manipulate the flow-subsequences more easily, we associate a set of numbers, P_j, with each node v_j, one value for each of the flow-subsequences in which v_j appears (see Appendix B and the following examples). To accomplish this, initialize i to 1 and repeat the following procedure for each node v_j. For each flow-subsequence b of B, if v_j is a node of b, include i in P_j and increment i. Using the results of this process, the flow-subsequences are expressed as sets of numbers. Next, the nodes are allotted weights determined by the current implementation of

the algorithm. Each flow-subsequence and flow-sequence is then assigned a weight which is the sum of the weights of its constituent flow-subsequences. Once this assignment has been made, the cost probabilities can be calculated.

3.3 Hierarchy Of Isomorphic Flow-Sequences

The flow-sequences are now ordered as a hierarchy of families of flow-sequences, F_i, where $b(F_i)$ is the lowest, by weight, member of the family F_i, q the number of families constructed so far, and s the index of the family to which the current flow-sequence will be adjoined.

Step 0: Initialization Step

Set q to 1, and F_1 to the flow-sequence of highest weight, ties resolved arbitrarily.

Step 1: If any flow-sequences remain, set h to the one of highest weight, set i to 1 and go to step 2. Otherwise, go to step 5.

Step 2: If $B(h)$ is a subset of $B(b(F_i))$, then set diff to $w(b(F_i))-w(h)$, set s to i, increment i and go to step 3. If $B(h)$ is not a subset of $B(b(F_i))$, increment i; if $i>q$, set s to i and go to step 4; otherwise go to step 2.

Step 3: If $i\leq q$, if $B(h)$ is a subset of $B(b(F_i))$, and diff is greater than $w(b(F_i))-w(h)$, then set diff to this new difference, and set s to i; if $i\neq q$, increment i and go to step 3; otherwise, step 4.

Step 4: Adjoin h to F_s, set q to i and return to step 1.

Step 5: Terminate.

The above procedure establishes families of isomorphic flow-sequences, where such flow-sequences are reduced. In each family, the flow-sequences are ordered by weight, the most costly being assigned the highest order (see Appendix B). The use of the hierarchy makes the removal of isomorphic copies of flow-sequences a simple process.

3.4 Reduction and Isomorphism of Flow-sequences

For each d, the pseudograph $G_d(X)$ is produced, and A_d computed. Then, the following processes are applied so that the flow-sequences can be reduced and isomorphic copies removed. First, A_d is considered for reduction. Within each flow-sequence, if any flow-subsequence appears more than once, only a single copy is retained in the walk; however, the total weight of the flow-sequence remains unchanged. Next, the reduced flow-sequences are checked for isomorphism. Each reduced flow-sequence, a', has an associated family, F', in the hierarchy. If there is a flow-sequence in both F' and A_d which has a higher order than a' in F', or a' occurs more than once in A_d, remove a copy of a' from A_d. Then, add w(a') to that flow-sequence in both A_d and F' which has the least order greater than or equal to the order of a'. This process yields the subset A_d' of A_d consisting of only reduced, non-isomorphic flow-sequences (see Appendix B and the examples in section 3.7).

3.5 Entropy, Conditional Entropy, Information Value

Using A'_d, the entropy, conditional entropy and information value are calculated for each fixed d in D(X). Then, associating a relative frequency with each d, the entropy, conditional entropy and information value are obtained by averaging over D(X). Each distinct implementation of the algorithm yields another set of costs for B and hence, defines a new input probability distribution. The evaluation and analysis of the algorithm which is now possible, is discussed with examples, in the remaining sections.

3.6 Further Analysis of the Information Value

In this study, the information value has been used for two purposes. First, to select from a set of algorithms, that one which is, in a cost sense, the most suitable for the given task, and secondly, to choose an implementation of that algorithm which is most appropriate in such a situation. In the following analysis, we assume that the flow-sequences have been reduced, and isomorphic copies removed. Also, we define an index set on the flow-subsequences b_j of a flow-sequence a by

$$I(a) = \{ j | b_j \text{ is in } B(a) \}.$$

3.6.1 Selecting An Algorithm

First, we assume that the algorithms are cost comparable within three or four per cent. For such algorithms, since the costs are more or less the same, the flow structure is the

dominant component of the information value; we obtain the largest information value from the structurally optimum algorithm. Specifically, let X and Y be two cost comparable algorithms. Furthermore, suppose that algorithm X has a distinctly better flow structure than algorithm Y, in the sense that each flow-sequence c of X consists of flow-subsequences not occuring in other flow-sequences of X. Then the information value indicates that algorithm X has a flow structure which can be used to point to a more efficient coding of the algorithm. To see this, we study each algorithm's information value. First analysing algorithm X, we note that in most cases the flow-subsequences b_j of a flow-sequence c_k do not appear in many other flow-sequences, so the conditional probability

$$P(c_k | b_j) = w(c_k)/w(c^j),$$

is close to unity and, thus, over $I(c_k)$, $P(c_k)$ is approximately equal to $\sum P(b_j)$; this shows that the walk probabilities are fairly uniformly distributed. However in the second algorithm this does not hold. Most flow-subsequences u_i of flow-sequence e_f of Y, occur in other flow-sequences. As a result, there are more flow-sequences for Y than for X. Thus, when calculating

$$P(e_f | u_i) = w(e_f)/w(e^i),$$

its value is considerably less than unity and hence the resulting flow-sequence probabilities

$$P(e_f) = \sum P(e_f | u_i) P(u_i),$$

as well as corresponding entropy, are less than those for algorithm X. Hence, since the flow-sequence weights are nearly

the same, the information value for algorithm X is greater than that of algorithm Y. Thus, when comparing unstructured algorithms, the one with the higher information value is selected, and an implementation of it which reflects the flow structure, coded. To summarize, when the costs for algorithms are comparable, the maximum information value indicates which algorithm is optimal relative to this factor.

3.6.2 Selecting the Algorithm Implementation

Having chosen the algorithm most suitable for the given task, an analysis of it, based on cost, is made. In this instance, the implementation with minimum cost is desirable; accordingly the minimum of the information values points to such an implementation. To see this, we consider the two ways in which the algorithm costs can be decreased.

Case 1: Some flow-sequence a* is made more efficient by applying optimizing techniques to its flow-subsequences. For simplicity, we assume that a* consists of flow-subsequences b_j which do not appear in any other flow-sequence of the algorithm, and that $P(a^*) < 0.5$. Then, if a* is coded more efficiently, i.e. $w(a^*)$ decreases, the information value also decreases. The proof follows.

Under the assumptions given above, let z_j be the the decrease in N_j, (z_j is zero if b_j is not in $B(a^*)$), $z = \Sigma z_j$, and

$N^* = \sum N_j$, where the sums are over $I(a^*)$. Recall that N is defined by $\sum w(b_j)$, where the sum is taken over B. Then the following equality holds for those j in $I(a^*)$.

$$\begin{aligned} P(a^*|b_j) &= w(a^*)/w(a^j) \\ &= w(a^*)/w(a^*) \\ &= 1 . \end{aligned}$$

Also, $\sum P(b_j)$ over $I(a^*)$ decreases, for if $\sum P(b_j)$ were to increase, then

$$\sum (N_j - z_j)/(N-z) \; > \; \sum (N_j/N).$$

This implies $(N^*-z)/(N-z) > N^*/N$, and hence $N < N^*$, a contradiction. So $\sum P(b_j)$ over $I(a^*)$ decreases, which implies $P(a^*)$ decreases, and hence, that $-P(a^*)\log P(a^*)$ decreases. Now $w(a^*)$ decreases by assumption, so

$$w(a^*)(-P(a^*)\log P(a^*)),$$

the information value of a*, decreases. Qed.

Now we examine the effect of decreasing the cost of some flow-subsequence b* which appears in many flow-sequences. This situation can be examined as two subcases.

Case 2-1: First suppose that b* is not in either $B(a_k)$ or $B(a^j)$, j in $I(a_k)$, and let z be the decrease in $w(b^*)$. Now $P(b_j)=N_j/N$ increases for b_j in $B(a_k)$ since N_j is fixed and N decreases. Also $P(a_k|b_j) = w(a_k)/w(a^j)$ is fixed, since both $w(a_k)$ and $w(a^j)$ are constant. Thus,

$$P(a_k) = \sum_{j=1}^{n} P(a_k|b_j)\; P(b_j)$$

increases, and hence $-P(a_k)\log P(a_k)$ increases. This

implies $-w(a_K)P(a_K)\log P(a_K)$ increases, since $P(a_K) < 0.5$ and $w(a_K)$ is fixed. That is, the information value increases.

Now, assume that b* is in $B(a^j)$, for some j in $I(a_K)$, but b* is not in $B(a_K)$. Thus $w(a^j)$ decreases, $P(a_K|b_j) = w(a_K)/w(a^j)$ increases, and, as above $P(b_j)$ increases. Hence, $P(a_K)$, and thus the information value of a_K, increases.

Thus, when b* is not in $B(a_K)$, the information value indicates that the change made was not beneficial to the flow-sequence a_K. If this is true for most flow-sequences, the overall information value increases, indicating that such a change should not be made.

Case 2-2: b* is in $B(a_K)$. Again let z be the decrease in w(b*). Let p' be the new $P(a_K)$, p the old $P(a_K)$, and $w'(a_K) = w(a_K)-z$. If $p' < p$, then $-w'(a_K)(p'\log p')$ is less than $-w(a_K)(p \log p)$. Thus, the information value decreases.

On the other hand, if $p' > p$, the information value still decreases for most z. The difference, p'-p, must be small, since even large z implies only small increases in $P(a_K|b_j)$ and small changes in $P(b_j)$, j in $I(a_K)$. We observe that the information value decreases

iff $w'(a_K)(-p'\log p') < w(a_K)(-p \log p)$,

iff $w(a_K)[(-p'\log p')-(-p \log p)] < z(p'\log p')$

iff $w(a_K)\left(\log\left(p'^{p'}/p^{p}\right)\right) < z(p'\log p')$.

And, since for most z, $\log\left(p'^{p'}/p^{p}\right)$ is approximately 0, this inequality is satisfied. That is, the information value decreases.

The usefulness of the information measure for selecting the most appropriate implementation of an algorithm is thus evident. When the implementation is an improvement, the corresponding information value decreases, and when this is not the case, the information value increases.

Some of the applications of such a measure are now given. First, if the probability of a flow-sequence is relatively large, but its weight is comparable to other flow-sequences, then the algorithm can be coded to reflect this flow, and hence to decrease execution costs. Also, when attempting to partition a program into segments, a flow-sequence with high probability can be informative. Such probability indicates that the flow-sequence has little interaction with other parts of the program, a major factor in a segmentation problem. Thirdly, if the conditional probabilities on the flow-sequences, relative to some flow-subsequence b, are consistently low, then such a flow-subsequence appears many times, and making it more efficient is reflected throughout the program.

To illustrate the two purposes the information measure is used for in this study, the following examples are included.

3.7 Examples

3.7.0 The Form and Purpose of the Examples

In order to analyse the selected algorithms, certain programs are necessary to produce and evaluate the output and calculate the information value. The language in which these

routines are written is ALGOLW. However, the programs representing the algorithms are expressed in the hypothetical assembly language, described in Appendix A.

To apply the measure, a task and algorithms to compute the task, are selected. The sorting of n numbers (equivalently the indices of n records) into order is chosen since several well documented algorithms for sorting exist, and the corresponding data sets are well defined (i.e. all possible orderings of n numbers). The two sort algorithms, 'heap' and 'merge' [21], are used as the possible candidates for the task of placing in order five numbers. In the following discussion, the five factorial elements of the data set are arbitrarily numbered from 1 to 120; we will refer to certain of these data in the examples given below. On this data set, the merge algorithm is not amenable to structuring since no reduction of flow-subsequences is possible. Although this latter point ordinarily implies that code optimization of the flow-sequences would be beneficial, the simplicity of the calculations, and the lack of interaction among the nodes within a flow-subsequence allow for no noticable improvements. As a contrast, the heap algorithm is studied. In this case, the 'shortcomings' of the merge algorithm are absent, and hence, both structuring and code optimization can be applied.

The task evaluated here is obviously somewhat trivial, so the concept of structuring is not as dominant as it would be in a more complex situation. For a task involving more

calculations and decisions, the use of subroutines as structures would be effective. However, the simplicity of the examples still allows the usefulness of this measure to be illustrated, both as a cost and structural complexity standard, and as an improvement on a measure based on execution costs alone.

When analysing an algorithm, the standard implementation is coded first. So at this time, no attempt is made to optimize any particular area of the program. Both the merge and heap algorithms are implemented in this manner. In addition, based on the results of an analysis of the information values, another implementation of the heap algorithm is given. In the examples, the execution costs associated with each implementation are included with the information value, so that a comparison can be made with the conventional measure. Table entries listed as costs, are in execution units, while the information value and conditional entropy are unitless.

The measure is first applied to the problem of deciding which of two algorithms should be used for a given task. The information value points to the algorithm which is more suitable for the situation. Moreover, once the algorithm has been selected, the measure further analyses it by providing information on its overall cost and structure, which indicates where attention should be focused in order to produce a more efficient implementation of the algorithm.

3.7.1 Comparison of Two Algorithms for the Same Task

In this section the first application of the measure is discussed. The heap algorithm, (H-I), as mentioned above, is amenable to structuring, while merge, (M), is not. The standard implementations are compared and consequently H-II, an implementation resulting from a structuring which improves the efficiency of heap, is included in the study.

In the first example, assuming the relative frequencies of the orderings are equal, the average costs of the algorithms are compared (see Table I). Merge is seen to be cheaper; however, removing the extreme cases (i.e. those with very high or very low costs), the costs are relatively comparable. Also, the number of nodes and cost per node are very close. Thus the information value can be used to evaluate the two algorithms, indicating that heap is more informative, i.e. given proper structuring the overall cost of heap, for this situation, would be less costly. The implementation of a single structural improvement (optimization of a flow-subsequence), reduces the average cost of heap below that of merge. Using the measure of average cost, it is unlikely that the heap algorithm would have been given further consideration for this particular application. In the next section a further analysis of heap is given.

Alg	Av Cost	Info Val
merge	198.75	35.195
heap-I	228.31	38.365
heap-II	198.18	37.608

Table I: Example 1

The next two examples involve particular data and their resulting output from the two algorithms. Such analysis is useful if a particular datum or type of datum is highly probable. In such a case, the algorithm that performs in the best way for the given datum is the one selected.

Datum	IV H-I	IV M	M cost	H-I cost	H-II cost
45	31.495	34.940	200.1	206.5	202.7
69	33.480	33.328	202.9	205.9	201.9
44	40.420	40.075	199.5	200.6	195.7

Table II: Examples 2 and 3

As shown in Table II in the entry for datum 45, the information value of merge is higher, indicating that even with proper structuring and optimizing it is unlikely that the heap algorithm will perform better than merge. The costs of H-I and H-II confirm this.

For each of data 69 and 44 the information values of the two algorithms are relatively close, but heap is slightly higher. This indicates that optimization methods should reduce the cost of the heap algorithm. Again the costs associated with H-I and H-II support this observation.

3.7.2 Analysis of Heap Algorithm

In the remaining examples, the flow-subsequences are expressed as sequences of numbers, corresponding to nodes, separated by dashes; parentheses indicate repeated flow-subsequences which are eliminated in the reduction process. Execution over each datum d generates five flow-sequences; those marked with an asterisk are isomorphic to others in the list. Only the set of non-isomorphic flow-sequences are considered in computing the information value. The lists of flow-subsequences and flow-sequences are given in Appendix B. The initialization nodes are omitted, since no optimization techniques are applied to these nodes.

The following example illustrates the use of the conditional entropy as a means of partitioning the data set according to structure. Within 'partitions', the improvements in cost resulting from a given change in the implementation of the algorithm are fairly close. When the conditional entropies differ, this indicates that the corresponding structures of the outputs do also. The smaller the conditional entropy, the more amenable to node or flow-subsequence improvements are the

sections of the algorithm associated with this datum. Consider the data pairs shown in Table III.

Datum	Info Val	Cond Ent	H-I cost	H-II cost	|I-II|
33	56.325	1.374	229.1	223.1	6.0
36	54.939	1.731	219.0	214.2	4.8
19	42.802	2.095	218.4	213.4	5.0
20	44.188	1.731	228.5	222.3	6.2

Table III: Example 4

The output flow-sequences for the above data are:

(33)

1* 1-3 7-11-16-21 27-28

1 1-3 7-11-16-21 (7-11-16-21) 27-28

2 2-4-3 7-11-16-21 9-15-23 27-28

2* 2-4-3 7-11-16-21 27-28

2* 2-4-3 9-15-23 27-28

(36)

1 1-3 7-11-16-21 27-28

2 1-3 7-11-16-21 5-12-19-25

3 2-4-3 7-11-16-21 9-15-23 27-28

3* 2-4-3 7-11-16-21 27-28

3* 2-4-3 9-15-23 27-28

(19)

1	1-3 7-11-16-21 27-28
2	1-3 7-11-16-21 5-12-19-25
3	2-4-3 7-11-16-21 9-15-23 27-28
4	2-4-3 8-13-17-22 27-28
3*	2-4-3 9-15-23 27-28

(20)

1*	1-3 7-11-16-21 27-28
1	1-3 7-11-16-21 (7-11-16-21) 27-28
2	2-4-3 7-11-16-21 9-15-23 27-28
3	2-4-3 8-13-17-22 27-28
2*	2-4-3 9-15-23 27-28

The information value of (33) is slightly higher than that of (36) due to the simpler structure and more costly flow-sequences of (33). This same observation holds for the second pair of data, with (20) having the higher information value. In both pairs, the lower conditional entropy indicates which datum has the simpler structure in its output. An implementation which reflects such a structure yields greater cost savings for the datum with the lower conditional entropy. Such an analysis is useful if it is known that a high portion of the data is of the form of either (19) or (20) say, and it is desirable to know whether optimization methods should be applied to the path followed by (19) or by (20). This measure indicates that by concentrating on (20) more overall savings can be obtained.

In the next example, the notion of partition and the effect of costs on the measure are discussed. The conditional entropy of both (3) and (5) are close (see Table IV), but their information values differ. From this, it is known that their structures are similar, but that (3) either has more costly flow-subsequences, or has flow-sequences which can be coded more efficiently to reflect their unique flow structures. This example illustrates that within 'partitions' the information value ranks the data according to the corresponding costs i.e. the higher the cost, the higher the information value.

Datum	Info Val	Cond Ent	H-I cost
3	53.334	1.610	228.5
5	35.287	1.611	204.7

Table IV: Example 5

The associated flow-sequences are:

(3)

1 1-3 8-13-17-22 27-28

2 1-3 7-11-16-21 (7-11-16-21) 27-28

3 2-4-3 7-11-16-21 9-15-23 27-28

3* 2-4-3 7-11-16-21 27-28

3* 2-4-3 9-15-23 27-28

(5)

1 1-3 8-13-17-22 27-28

2 1-3 7-11-16-21 (7-11-16-21) 27-28

3 2-4-3 7-11-16-21 10-18-26

4 2-4-3 7-11-16-21 27-28

3* 2-4-3 10-18-26

The third example of this section (see Table V) shows that if a datum has a much higher information value than another, and its conditional entropy is lower, then this datum, as well as consisting of more costly flow-subsequences, is more amenable to node or flow-subsequence improvement. Thus, by selecting the more informative of equally probable data, and applying optimizing techniques to its flow path, greater cost reductions result than if such processes were applied to the other datum.

Datum	Info Val	Cond Ent	H-I cost	H-II cost	|I-II|
11	54.291	1.887	216.3	211.6	4.7
12	67.106	1.715	228.2	222.3	5.9

Table V: Example 6

The corresponding flow-sequences are listed below.

(11)

1*	1-3 7-11-16-21 27-28
1	1-3 7-11-16-21 8-13-17-22 27-28
2	2-4-3 7-11-16-21 9-15-23 27-28
2*	2-4-3 7-11-16-21 27-28
3	2-4-3 10-18-26

(12)

1*	1-3 7-11-16-21 27-28
1	1-3 7-11-16-21 8-13-17-22 27-28
2	2-4-3 7-11-16-21 9-15-23 27-28
2*	2-4-3 7-11-16-21 27-28
2*	2-4-3 9-15-23 27-28

In this instance (12) is seen to have the simpler structure, lacking the extra flow-sequence 10-18-26 which appears in (11). Here flow-subsequence 27-28 was optimized.

In the last example a case where the conditional entropies are approximately the same is considered. This implies that the outputs corresponding to the two data have nearly the same structure. Sample (49) has a somewhat costlier output, a fact reflected in the information value given in Table VI. However, since the conditional entropies indicate similar structures, the marginal difference in costs will not cause marked

dissimilarities in the cost savings. That is, the flow-sequences of either datum can be structured to increase the efficiency, and the resulting cost decreases for both cases will be nearly the same. Due to the simplicity of the sample task, the flow-subsequences of (49) and (57) are fairly similar; however, this is not necessary in order that the conditional entropies be the same.

Datum	Info Val	Cond Ent	H-I cost	H-II cost	|I-II|
49	36.686	1.820	198.1	193.1	5.0
57	35.450	1.828	195.1	189.7	5.4

Table VI: Example 7

The flow-sequences associated with the table entries are given below.

(49)

1 1-3 7-11-16-21 27-28

2 1-3 5-12-19-25

3 2-4-3 7-11-16-21 9-15-23 27-28

4 2-4-3 8-13-17-22 27-28

3* 2-4-3 9-15-23 27-28

(57)

1*	1-3 8-13-17-22 27-28
1	1-3 8-13-17-22 27-28
2	2-4-3 7-11-16-21 10-18-26
3	2-4-3 8-13-17-22 27-28
4	2-4-3 9-15-23 27-28

3.7.3 Summary Of The Examples

The above examples are intended to demonstarate how the information value, together with the conditional entropy, can be used to aid in the analysis of an algorithm. The measure can point to expensive data (i.e. costly for algorithms to execute), but as well, can indicate which paths through the algorithm shcuild be considered for code optimization in an attempt to obtain the maximum cost saving. In more complex examples, improvements on nodes or flow-sequences rather than just flow-subsequences would be in order.

The advantages of using the information value as a complexity measure are evident from these examples.

3.8 Summary

In this thesis, an attempt was made to define a cost and structural complexity measure for an algorithm. To accomplish this, we defined an information theoretic model of the execution of an algorithm, in which the input is a set of subwalks, and the output certain walks, of a graph theoretic representation of the algorithm. Cost is included in the model through the definition of a cost probability scheme, and structure through the concepts of reduction and isomorphism. An information value for each implementation of the algorithm is calculated. It is shown that this value provides all the information that the conventional measure of cost alone does. Moreover, it presents structural information which indicates the amount of interaction between program sections, and points to dominant, repeated and independent flow patterns, and to structural similarities. Under the assumption of comparable costs, the maximum information value points to a structurally optimum algorithm; when the structure is fixed, i.e. analysing a given algorithm, the minimum cost implementation has the smallest information value. The appropriateness of these considerations in the analysis of an algorithm has been demonstrated in Chapter III. More generally, this study has demonstrated the feasibility of using information theory to measure the complexity of an algorithm.

We conclude with some suggestions for improving the model treated here. First, the introduction of more structural

parameters may improve the model. Presently, reduction and isomorphism have proved beneficial in evaluating an algorithm; however, a redefinition or expansion of these may yield a more informative measure. The model obtains the information value from the weighted average of the entropy less the conditional entropy. Without the inclusion of these weights, the measure becomes much more responsive to structural information, and less sensitive to costs. In certain instances this may yield a more valuable measure than the one defined in this study. Of note, when using the 'unweighted' measure, is the fact that the most appropriate implementation has the largest information value. The inclusion of other parameters in the model might also prove useful, but this would increase the cost of applying the measure when this cost may already seem prohibitive. However, as with most studies concerned with the complexity of algorithms, this analysis is based on the following assumption. The task which the algorithm or algorithms under consideration will be computing is central to some process that is to be repeated many times (for instance in a business application, some procedure which must be calculated daily). Thus, the cost in performing a complexity analysis for such a task may well be neglible relative to the overall savings incurred through the use of the appropriate algorithm and its optimal implementation.

BIBLIOGRAPHY

1. AHO, A. and ULLMAN, J., Optimization of Straight Line Programs, SIAM Journal Comp. 1, (1972) pp. 1-19

2. ALEKSEEV, V.E., Sorting Algorithms with Minimum Memory, Cybernetics, 5, (1969), pp. 642-648.

3. ALLEN, F.E., Control Flow Analysis, Proc. of a Symposium on Compiler Optimization, SIGPLAN Notices, ACM, New York, July 1970, pp. 1-19.

4. ALLEN, F.E., Program Optimization, in: Annual Review in Automatic Programming Vol. 5, Pergamon Press, New York, 1969.

5. ASH, R. Information Theory, Wiley Interscience, New York, 1965.

6. BACHMAN, P., A Contribution to the Problems of the Optimization of Programs, Inform. Processing '71 North-Holland, Amsterdam, 1972, pp. 397-401.

7. BAKU, S. On Reduction of Program Schemes, SIAM J. Comp. 16, (1968) pp. 328-339.

8. BERZTISS, A., A Note on Segmentation of Computer Programs, Inform. Control 12, (1968) pp. 21-22.

9. BEUS, H., The Use of Information in Sorting J.A.C.M. 17, (1970), pp. 482-495.

10. COCKE, J., Global Common Subexpression Elimination, Proceedings of a Symp. on Compiler Optimization, SIGPLAN Notices, ACM, New York, July 1970, pp. 20-24.

11. FRAZER, W.E., Analysis of Combinatory Algorithms - A Sample of Current Methodology, AFIPS Conf. Proceedings, Spring Joint Computer Conference AFIPS Press, Montvale, N.J., 1972, pp. 483-491

12. GALLAGER, R., Information Theory and Reliable Communication, Wiley, New York, 1968.

13. GREEN, C., A Path Entropy Function For Rooted Trees, J.A.C.M. 20, (1973), pp. 378-384.

14. HARARY, F., Graph Theory, Addison-Wesley, Reading Mass., 1969.

15. HARTMANIS, J., Computational Complexity of Random Access Stored Program Machines, Math. Systems Theory 5, (1971) pp. 232-245.

16. HOPKINS, M., An Optimizing Compiler Design, Inform.

Processing '71, North-Holland, Amsterdam, 1972, pp. 391-396.

17. IANOV, I., On the Equivalence and Transformation of Program Schemes, Comm. A.C.M. 1, (1958), pp. 8-12.

18. IANOV, I., On Matrix Program Schemes, Comm. A.C.M. 1, (1958), pp. 3-6.

19. KARP, R., A Note on the Application of Graph Theory to Digital Programming, Inform. Control 3, (1960), pp. 179-190.

20. KARREMAN, G., Topological Information Content and Chemical Reactions, Bull. Math. Biophys. 17 (1955), pp. 279-285.

21. KNUTH, D., Sorting and Searching, The Art of Computer Programming, Vol. 3, Addison-Wesley, Reading, Mass., 1973.

22. KRIDER, L., A Flow Analysis Algorithm, J.A.C.M. 11, (1964), pp. 429-436.

23. MOWSHOWITZ, A., Entropy and the Complexity of Graphs, Doctoral Dissertation, University of Michigan, 1967.

24. NIEVERGELT, J., On the Automatic Simplification of Computer Programs, Comm. A.C.M. 8, (1965), pp. 366-370.

25. PATERSON, M., Program Schemata in: Machine Intelligence Vol. 3, (D. Michie, editor), American Elsevier, New York, 1968.

26. PICARD, C., Theorie des Questionnaires, Gauthier-Villars, Paris, 1965.

27. RASHEVSKY, N., Life, Information Theory, and Topology, Bull. Math. Biophys. 17 (1955), pp. 229-235.

28. RUTLEDGE, J., On Ianov's Program Schemata, J.A.C.M. 11, (1964), pp. 1-9.

29. SCHURMANN, A., The Application of Graphs to the Analysis of Distribution of Loops in a Program, Inform. Control 7, pp. 275-282.

30. TRUCCO, E., A Note on the Information Content of Graphs, Bull. Math. Biophys. 18 (1956), pp. 129-135.

31. WARSHALL, S., On Computational Cost, in: Annual Reviuew in Automatic Programming Vol. 5, Pergamon Press, New York, 1969.

32. WOODGER, M., On Semantic Levels in Programming, in: Info. Processing '71, North-Holland, Amsterdam, 1972, pp. 402-407.

Appendix A

Here we give the instruction set of the hypothetical assembly language in which the algorithms are 'written'. These instructions are very basic to insure they remain machine independent. The execution costs are based on the MIX [21], PDP-10, and CDC assembler language timings. The following assumptions about this language are made.

(1) there are 8 registers, which are in fast memory and can be used for indexing.

(2) there is one accumulator.

(3) input and output are ignored.

The costs listed are in execution units; costs in parentheses are in effect if indexing is used. In the statement of the instruction, capital letters refer to memory locations, small letters to registers. C(x) is the contents of location or register x; Acc refers to the accumulator.

INSTRUCTION		COST
JUMP-	unconditional jump	1.2 (1.3)
JUMPE	jump if Acc = 0	
JUMPLE	jump if Acc $\leq$ 0	
JUMPL	jump if Acc < 0	
JUMPG	jump if Acc> 0	
JUMPGE	jump if Acc $\geq$ 0	
JUMPNE	jump if Acc = 0	

SETZM	A	set C(A) to 0	1.7 (1.8)
SETZR	,i	set C(i) to 0	1.0 (1.1)
SETR	j,i	set C(j) to C(i)	1.4
SETIR	i,n	set C(i) to n	1.0 (1.1)
SETMR	A,i	set C(i) to C(A)	1.6 (1.8)
SETRM	A,i	set C(A) to C(i)	1.8 (1.9)
SETNM	A	set C(A) to -C(A)	1.7 (1.9)
SETNR	,i	set C(i) to -C(i)	1.5 (1.7)
ADD)R	i,j	add C(i) to C(j)	1.6 (1.8)
SUB)I	A,n	add n to C(i)	1.2 (1.3)
MR	A,i	add C(A) to C(i)	2.2 (2.3)
DIVI	,n	divide Acc by n	11.3
CMPZM	A	C(A) - 0; set flag	1.7 (1.9)
CMPZR	,i	C(i) - 0; set flag	1.5 (1.7)
CMPR	i,j	C(i) - C(j); set flag	1.6 (1.8)
CMPI	,n	C(Acc) - n; set flag	1.2 (1.3)
CMPM	A	C(Acc) - C(A); set flag	1.9 (2.0)
BLT	n	move n contiguous words of memory	0.8+(2.1) n

Appendix B

In order to illustrate some of the concepts introduced in this thesis, we will present a partial analysis of H, the heap algorithm [21]. We first give the algorithm in its textual form. Based on this, we produce G(H), the graph of H. Then, so that the analysis programs have use of a numeric representation, the sets P_j are established. Finally we obtain the numeric representation of the flow-subsequences and flow-sequences, where the latter is listed in family notation (see procedure in section 3.3).

Heapsort: A file of numbers $R_1, R_2, \ldots, R_N$ is a 'heap' if

$$R_{[j/2]} \geq R_j \quad \text{for} \quad 1 \leq [j/2] < j \leq N,$$

where [x] is the greatest integer in x. Thus, $R_1 \geq R_2$, $R_1 \geq R_3$, $R_2 \geq R_4$, etc., and this implies that the largest number appears 'on top of the heap',

$$R_1 = \max(R_1, R_2, \ldots, R_N).$$

If an arbitrary input file is transformed into a heap, a 'top-down' selection procedure can be used to sort.

Algorithm H. Numbers $R_1, \ldots, R_N$ are rearranged so that after sorting is complete, they will be in order. First the file is rearranged so that it forms a heap, then the top of the heap is repeatedly removed and transferred to its proper final position. Assume that $N \geq 2$.

H1.[Initialize] Set d to [N/2] + 1, r to N.

H2.[Decrease d or r.] If d>1, set d to d-1, and R to R_d. (If d>1, the file is being made into a heap; if d=1 then the file is already a heap.)

H2b. Otherwise set R to R_r, R_r to R_1, and r to r-1; if this makes r=1, set R_1 to R and terminate.

H3.[Prepare for 'sift-up'] Set j to d. (At this point we have

$$R_{[j/2]} \geq R_j \quad \text{for} \quad d < [j/2] < j \leq r;$$

and R_K is in its final position for $r < k \leq N$.

H4.[Advance downward.] Set i to j and j to 2j. (In the following steps we have $i = [j/2]$.) If j<r, go to step H5; if j=r, go to step H6; and if j>r, go to H8

H5.[Find 'larger' son.] If $R_j < R_{j+1}$, then

H5b: set j to j plus 1.

H6.[Larger than R] If $R \geq R_j$, then go to step H8.

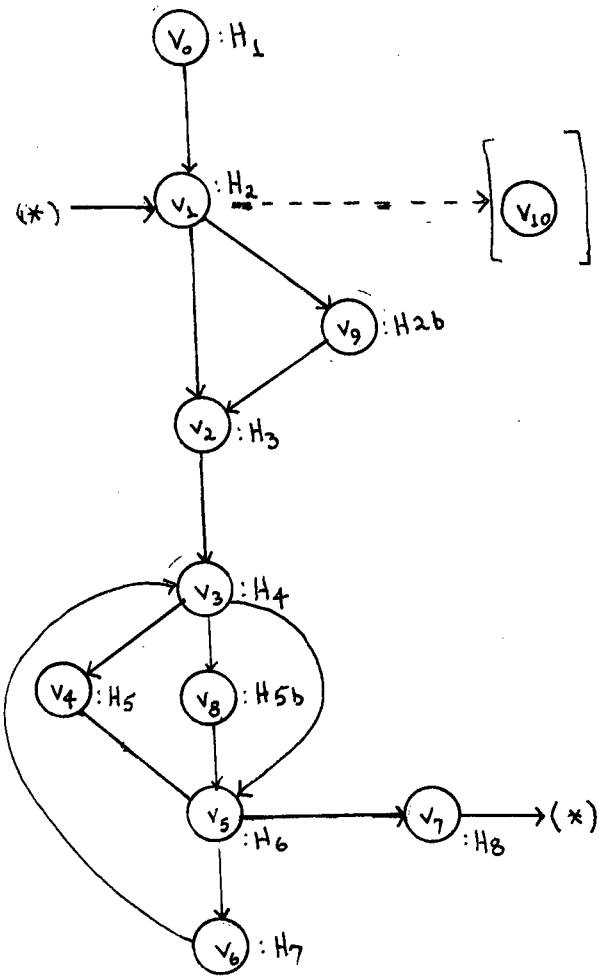
H7.[Move it up.] Set R_i to R_j, and go back to step H4.

H8.[Store R.] Set R_i to R. (This terminates the 'sift-up' algorithm initiated in step H3.) Return to step H2.

Graph of Algorithm H

$V_s = V_1$

$V_t = V_{10}$



G (H)

j of Node X_j	P_j
1	1,2 [t=10]
2	3
3	5,6,7,8,9,10,27
4	11,12
5	15,16,17,18,19,20
6	21,22,23
7	24,25,26,28
8	13,14
9	4

Flow-Subsequences

1-3

8-13-17-22

27-28

7-11-16-21

10-18-26

2-4-3

9-15-23

6-14-20-24

5-12-19-25

Families of Flow-Sequences		Costs
F_1:	2-4-3 7-11-16-21 10-18-26	44.7
	2-4-3 10-18-26	24.4
F_2:	1-3 7-11-16-21 8-13-17-22 27-28	56.6
	1-3 7-11-16-21 27-28	37.2
	1-3 8-13-17-22 27-28	36.6
F_3:	1-3 7-11-16-21 6-14-20-24	46.8
	1-3 6-14-20-24	26.5
F_4:	2-4-3 7-11-16-21 9-15-23 27-28	56.6
	2-4-3 7-11-16-21 27-28	41.5
	2-4-3 9-15-23 27-28	36.3
F_5:	1-3 7-11-16-21 5-12-19-25	47.4
	1-3 5-12-19-25	27.1
F_6:	2-4-3 8-13-17-22 27-28	40.9