INFORMATION TO USERS

This material was produced from a microfilm copy of the original document. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the original submitted.

The following explanation of techniques is provided to help you understand markings or patterns which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting thru an image and duplicating adjacent pages to insure you complete continuity.

2. When an image on the film is obliterated with a large round black mark, it is an indication that the photographer suspected that the copy may have moved during exposure and thus cause a blurred image. You will find a good image of the page in the adjacent frame.

3. When a map, drawing or chart, etc., was part of the material being photographed the photographer followed a definite method in "sectioning" the material. It is customary to begin photoing at the upper left hand corner of a large sheet and to continue photoing from left to right in equal sections with a small overlap. If necessary, sectioning is continued again - beginning below the first row and continuing on until complete.

4. The majority of users indicate that the textual content is of greatest value, however, a somewhat higher quality reproduction could be made from "photographs" if essential to the understanding of the dissertation. Silver prints of "photographs" may be ordered at additional charge by writing the Order Department, giving the catalog number, title, author and specific pages you wish reproduced.

5. PLEASE NOTE: Some pages may have indistinct print. Filmed as received.

Xerox University Microfilms
300 North Zeeb Road
Ann Arbor, Michigan 48106
LONGE, Oluwumi, 1932-
AN INDEX OF SMOOTHNESS FOR COMPUTER PROGRAM FLOWGRAPHS.

The Ohio State University, Ph.D., 1974
Computer Science

Xerox University Microfilms, Ann Arbor, Michigan 48106

© 1975

OLUWUMI LONGE

ALL RIGHTS RESERVED

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED.
AN INDEX OF SMOOTHNESS
FOR COMPUTER PROGRAM FLOWGRAPHS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate
School of The Ohio State University

By

Oluwumi Longe, B. Sc., M. Sc.

* * * * *

The Ohio State University
1974

Reading Committee:
C. R. Foulk
H. W. Buttelmann
M. T. Liu

Approved By

C. R. Foulk
Adviser
Department of Computer
& Information Science
ACKNOWLEDGEMENTS

Financial support for this work was in the form of a fellowship awarded me by the Rockefeller Foundation of New York. This is gratefully acknowledged.

I thank the University of Ibadan, Nigeria for granting me an extended study leave to undertake this research.

I thank Dr. R. L. Wilson, Ohio Wesleyan University and Dr. M. C. Yovits, Chairman, Department of Computer and Information Science, Ohio State University, for their cooperation with the University of Ibadan, Nigeria and with me in the initiation of this project.

I am grateful to Dr. C. R. Fouk for his assistance in identifying the research topic and for his guidance and encouragement throughout the period of the research.

I thank Dr. H. W. Buttelmann and Dr. M. T. Liu, members of my reading committee for their helpful suggestions.

Discussions with Dr. L. J. White on some aspects of graph theory helped me in the clarification of some problems. I am grateful to him.

Thanks are due to Mrs. Donna Curfman who diligently carried out the typing chores.

My family have given me every support and sustenance throughout the period of this project. This final product of the project is therefore dedicated to them: my wife, Remi, my daughters, Kemi and Joke and my son, Tunji.
VITA

March 20, 1932 ..................................................... Born Ikere-Ekiti, Nigeria

1959 ..................................................... B. Sc., Physics,
University of London, England

1960 ..................................................... Principal, Lisabi College
Abeokuta, Nigeria

1961 - 1963 ...................................................... Senior Science Master,
Lagelu Grammar School,
Ibadan, Nigeria

1967 ..................................................... M. Sc., Computer Science,
Sydney University, Australia

1967 ..................................................... Operations Manager,
U. A. C. Computer Centre,
Lagos, Nigeria

1968 ..................................................... Director, Computing Centre,
University of Ibadan, Nigeria

PUBLICATIONS

"Reading Gapless Tape" with C. S. Wallace, IEEE Transactions EC-16,
pp 517-518, August 1967.

"Immunoglobulins, Transferin, Caeruloplasmin and Heterophile
Antibodies in Kwashioker" with H. McFarlane, S. Reddy, A. Cooke,

"Man and the Computer." The Nigerian Christian, 4, 11, pp 2-3,
November, 1970.
FIELDS OF STUDY

Major Field: Computer and Information Science


Studies in Programming Languages. Professors W. H. Buttleman, L. J. White and A. W. Biermann

Studies in Computer Architecture. Professor M. T. Liu
LIST OF TABLES

1 - 8 Variables of Algorithms ............. 85
9 Analysis by Programming Languages .... 93
10 Correlation of Index of Smoothness with other Variables (All Selected Flowgraphs) .... 94
11 Correlation of Index of Smoothness with other Variables (ALGOL Flowgraphs) ....... 95
12 Correlation of Index of Smoothness with other Variables (FORTRAN Flowgraphs) .... 96
13 Analysis by "age" of Algorithms ......... 97
14 ACM Regions, 1974 ..................... 98
15 Analysis by Regions of Origin of Algorithms ..................... 99
<table>
<thead>
<tr>
<th>LIST OF CHARTS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. General Form of a Flowgraph</td>
<td>6</td>
</tr>
<tr>
<td>II. Types of Nodes in a Flowgraph</td>
<td>7</td>
</tr>
<tr>
<td>III. A Standard Flowgraph and its Minimal Form</td>
<td>9</td>
</tr>
<tr>
<td>IV. Basic Control Elements</td>
<td>11</td>
</tr>
<tr>
<td>V. Two Specialized Types of Iteration</td>
<td>12</td>
</tr>
<tr>
<td>VI-VIII. The Productions of Grammar SG</td>
<td>14-17</td>
</tr>
<tr>
<td>IX. Decision Elements</td>
<td>21</td>
</tr>
<tr>
<td>X. Two Examples of Decision Elements</td>
<td>22</td>
</tr>
<tr>
<td>XI. General Form of a Smooth Cyclic Interval</td>
<td>29</td>
</tr>
<tr>
<td>XII. Generation of a Cyclic Interval by SG</td>
<td>30</td>
</tr>
<tr>
<td>XIII. Invalid Loop Exit</td>
<td>32</td>
</tr>
<tr>
<td>XIV. Deferred Interval Reduction</td>
<td>36</td>
</tr>
<tr>
<td>XV. A Non-Smooth Flowgraph</td>
<td>37</td>
</tr>
<tr>
<td>XVI. Path Splitting</td>
<td>42</td>
</tr>
<tr>
<td>XVII. A Non-Smooth Acyclic Interval</td>
<td>45</td>
</tr>
<tr>
<td>XVIII. Transformation of a Full Non-Smooth Acyclic Interval</td>
<td>46</td>
</tr>
<tr>
<td>XIX. Recovery of Transformed Flowgraph</td>
<td>47-49</td>
</tr>
<tr>
<td>XX. Subsets of the Nodes of a Full Cyclic Interval</td>
<td>51</td>
</tr>
<tr>
<td>XXI. Correcting for Invalid Loop-Exits</td>
<td>52</td>
</tr>
<tr>
<td>XXII. Two Examples of Highest-Order Flowgraphs which are not Deferred Reducible</td>
<td>56</td>
</tr>
<tr>
<td>XXIII. Corrections for Multiple Loop-Entries</td>
<td>63</td>
</tr>
<tr>
<td>XXIV. Calculation of Index of Smoothness</td>
<td>68</td>
</tr>
<tr>
<td>XXV. Flowgraph of Program &quot;SMOOTH&quot;</td>
<td>71-73</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

Computer programming has now been practised for more than a quarter-century. Within that time, programming techniques have evolved and programming languages have been developed. It is possible to write several programs in the same language or in different languages to compute the same function. Assuming that they all correctly compute the function, how can the "best" of them be selected? Two of the discriminating criteria currently used are:

a. Program length. The program which uses the least amount of memory space for the storage of program instructions and data is considered as the best.

b. Execution time. The program which runs to completion in the shortest time may be chosen as the best.

A combination of both criteria may be employed to make a selection.

More recently, interest has been shown in programs which have variously been described either as "decomposable" (1) or "intelligible" (2) or "more readable" (3) or "structured" (4) or "well-formed" (5) or "self-documenting" (6) or smooth (11). These terms relate, in a general sense to the structure of the program. They seem to imply that a desirable property of a program is its inherent quick revelation of its structure, i.e., how quickly its flow of control is revealed to a reader of the program.
Such structural simplicity is indeed very important. It is easier to trace the flow of control of a program if it is structurally simple even if its author or an explicit write-up is not available. To a group working on a big programming project, it is essential that the modules into which the project is divided be structurally simple and interface correctly. Debugging of a program is easier if it has simple structure. Optimizing compilers usually construct the control flowgraph of the program to be compiled in order to obtain the static and global relationships of expressions and to determine the locality of data. These tasks are facilitated if the program to be compiled has a simple structure.

No measure of this important property of programs currently exists.

Objectives of Research

The objectives of this study are as follows:

1. To define precisely a set of programs which possess this desired property of structural simplicity.

2. To develop an algorithm which determines whether a given program's flowgraph does or does not belong to the set.

3. To develop an algorithm which transforms a non-member to a member of the set.

4. To establish an index of measurement of structural simplicity.
Importance of Research

This study is an attempt to obtain a quantitative measure of the structure of programs' flowgraphs. Such a measure does not yet exist and it would be useful.

It gives a more tangible measure than those earlier mentioned vague terms, which are currently in use. It provides a numerical value which can be associated with program flowgraphs. Such quantification may lead to more objectivity in discussions of program structure.

The algorithms developed in the study may be useful in the areas of Optimization, Parallelism, and Program Verification.

Lowry and Medlock (7) expected that their work on Object Code Optimization would in the future be extended in several directions:

...The analysis required to optimize may be extended to detect irregularities in the control flow and data flow of programs.

Less immediate applications of program analysis will involve extensive restructuring of programs by automatic or semi-automatic means.

The writers feel that the area of analysis and transformation of programs is extremely fruitful for programming research.

The index of measurement established in this study may be used to:

select the "best" of a set of programs which compute the same function.

rank programs of students who are learning to write programs which have simple structures.
investigate any relationship between the structure of a program and other properties of the program.

Outline of the Rest of this Study

In Chapter 2, "smooth" programs are defined. Algorithms are developed to recognize smooth programs and transform non-smooth to smooth programs.

In Chapter 3, an index of smoothness is defined and its derivation is presented.

In Chapter 4, a random sample of seventy-eight programs were chosen from the algorithms published in the Communications of the Association for Computing Machinery (CACM). The flowgraphs of these programs are used as an experimental data base. The index of smoothness is computed for each flowgraph. A correlation analysis is then performed to determine whether there is any relationship between the index of smoothness and some other variables of a program.

In Chapter 5, the results are summarized and conclusions are drawn.
CHAPTER 2
SMOOTH PROGRAMS

In this chapter, we consider programs whose flows of control are explicitly representable in the form of a flowgraph.

We assume that each program is a proper program (8), i.e., there is exactly one flowgraph-entry arc and one flowgraph-exit arc and for every node in the flowgraph, there is at least one path from the flowgraph-entry arc through that node to the flowgraph-exit arc.

2.1 Program Flowgraph.

A general representation of a flowgraph is shown in Chart I. The START and STOP symbols are markers indicating respectively the beginning and end of the flowgraph. S represents the main structure of the flowgraph. The structure, S, is a connected, directed graph (digraph), which may be denoted by the pair (V,A) where V represents the set of nodes and A the set of directed arcs of the flowgraph.

2.1.1 Nodes of a flowgraph.

The set of nodes of a flowgraph consists of three subsets: computation nodes, decision nodes and collector nodes (8). Their representations are shown in Chart II, a,b, and c respectively. The computation node has exactly one immediate successor node. The decision node has exactly two immediate successor nodes. The collector node is always at the junction of two arcs which are directed towards the same node.
CHART I

General Form of a Flowgraph

START

S

STOP
CHART II

Types of Nodes in a Flowgraph

a

b

c
2.1.2 Standard Flowgraph.

A flowgraph is in standard form if every one of its nodes, except the exit node, has at least one immediate successor node or at most two immediate successor nodes. Conversely, every node of a standard flowgraph, except the entry node, has at least one immediate predecessor node or at most two immediate predecessor nodes. This definition makes the collector node a necessary element of a flowgraph and assigns to the collector node a non-trivial role, namely, that of ensuring that any node has no more than two immediate predecessors.

Let a "neighbor node" be defined as either an immediate predecessor or an immediate successor, then, any node of a standard flowgraph has no more than three neighbor nodes.

2.1.3 Minimal Flowgraph.

A minimal flowgraph is a standard flowgraph which does not contain consecutive computation nodes.

A flowgraph containing consecutive computation nodes can be reduced to its minimal form by coalescing each set of consecutive computation nodes into a single computation node.

Chart IIIb is the minimal form of Chart IIIa. The structure of a flowgraph is easier to analyze if the flowgraph is represented in its minimal form. The basic logic of the program can more easily be told at a glance if the flowgraph is in its minimal form.

In the rest of this study, a flowgraph is assumed to be a standard, minimal flowgraph unless otherwise specified.
CHART III

A Standard Flowgraph and its Minimal Form

---

Diagram a:

Diagram b:
2.1.4 Basic Control Elements.

The directed arcs of a flowgraph indicate how control flows from the program entry node. Control can flow from a node to a single immediate successor node. Control can diverge from a single node to more than one immediate successor. Control can flow forwards or backwards.

Three different types of basic control elements are shown in Chart IV a, b, c. These are, respectively, concatenation, alternation and iteration control elements. They represent some of the control constructs of some existing programming languages. The concatenation element represents implicit sequential control. The alternation element represents the "if-then-else" control and the iteration element represents "loop" control.

The iteration element, Chart IVc, represents a more general loop control than those commonly used by existing programming languages. If node N2 of the element is empty, then the iteration element of Chart Va is obtained. Chart Va is equivalent to the "do-until" loop control. Chart Vb is obtained if node N1 of Chart IVc is empty. Chart Vb is equivalent to the "do-while" loop control.

The set of flowgraphs which can be constructed using only the three control elements of Chart IV constitute an interesting set and a detailed investigation of that set is the subject of the rest of this chapter.
CHART IV

Basic Control Elements

a

b

c
CHART V

Two Specialized Types of Iteration

Diagram a

Diagram b
2.2 A Graph Grammar.

Consider a graph grammar, $SG$:

$$SG = (V_N, V_T, P, S)$$

$V_N = \{ S, C, A, I \}$, the set of non-terminal symbols;

$V_T = \{ \text{flowgraph symbols} \}$, the set of terminal symbols;

$P = \{ P_1, P_2, P_3, P_4 \}$, the set of productions;

$S$ is the start symbol.

The 14 productions, $P$, are shown in Charts VI, VII, and VIII.

The graph grammar $SG$ is similar to a context-free grammar.

The "language" generated by the grammar $SG$ is the set of all flowgraphs which can be constructed using only the three control elements of concatenation, alternation and iteration as building blocks.

Definition 2.1: A smooth flowgraph is a flowgraph which is generable by the graph grammar, $SG$, above.

Definition 2.2: A smooth program is a program whose flowgraph is smooth.

2.3 Recognition of Smooth Flowgraphs.

If a given flowgraph can be shown to be generable by the graph grammar, $SG$, then the flowgraph is smooth; otherwise, it is non-smooth. It is useful at this point to recall some of the parsing techniques in linguistics. A sentence is grammatical in $G$ if, starting with the terminal string (the sentence) and choosing appropriate productions of $G$, a progressive reduction into non-terminals and ultimately a reduction into the start symbol of $G$ is achieved. This is called a "bottom-up" parse of the sentence.
CHART VI

The Productions of Grammar SG

\[ P_{11}: S \rightarrow \]

\[ P_{12}: S \rightarrow C \]

\[ P_{13}: S \rightarrow A \]

\[ P_{14}: S \rightarrow I \]

\[ P_{2}: C \rightarrow \]

\[ S \rightarrow S \]
CHART VII

The Productions of Grammar SG

\[ P3.1: \quad A \Rightarrow S \]

\[ P3.2: \quad A \Rightarrow S \]

\[ P3.3: \quad A \Rightarrow S, S \]
CHART VIII

The Productions of Grammar SG

P4.1: \[ I \rightarrow \]

P4.2: \[ I \rightarrow \]

P4.3: \[ I \rightarrow \]
CHART VIII
The Productions of Grammar SG

$P_{44}: I \rightarrow$

$P_{45}: I \rightarrow$

$P_{46}: I \rightarrow$
A given flowgraph can be parsed, bottom-up as follows:

(i) identify subgraphs of the flowgraph such that each subgraph is generable by one or more of the productions of $SG$.

(ii) reduce each such subgraph to a single node and thus obtain a smaller, higher-order flowgraph.

(iii) repeat steps (i) and (ii) above as many times as possible.

(iv) if the highest-order flowgraph obtainable in this way is a single node, then, the given flowgraph is deferred reducible.

(v) if the given flowgraph is not deferred reducible, then, the parse of the flowgraph in grammar $SG$ is a failure. Therefore, the given flowgraph is not a "sentence" of $SG$, i.e., the flowgraph is non-smooth.

2.3.1 Intervals of a Flowgraph.

The bottom-up parse of a flowgraph as discussed in the previous section requires the identification of two terminal subgraphs of the flowgraph.

A graph construct, defined by Cocke (9) and named "interval," can be used to partition a flowgraph into subgraphs.

Definition 2.3: An interval is a maximal, single-entry subgraph, such that any cycle in the interval passes through the entry node (10).

An interval always has a single entry. However, it can have one or more than one exit. Therefore, an interval is not always a two-terminal subgraph.
Definition 2.4: If an interval has exactly one exit, then, it is a full interval; otherwise it is a partial interval.

Let $\Gamma$ be the immediate successor function. Let an interval be denoted by 
$$\{ h, n_1, n_2, \ldots, n_m \}$$
where $h$ is the entry (or header) node, $N_i$ ($1 \leq i \leq m$) is any other node in the interval. If there exists, in the interval, a node $N_k$ ($1 \leq k \leq m$) such that $\Gamma(N_k) = h$, then the interval contains a cycle and node $N_k$ is a latching node.

Definition 2.5: If an interval contains a cycle, then, it is a cyclic interval; otherwise, it is an acyclic interval.

An algorithm which can be used to partition a given flowgraph into its intervals has been defined by Allen (10).

Definition 2.6: A smooth interval is a full interval which is generable by the graph grammar, SG.

2.3.1.1 Full Acyclic Intervals.

A full acyclic interval is a two-terminal acyclic interval. If a full acyclic interval contains only one node, then, it is of the form $\begin{array}{c}
\text{\textbullet} \\
\downarrow \\
\text{\textbullet}
\end{array}$, otherwise, it contains one or more decision nodes.

Consider a full acyclic interval containing at least one decision node. Since there is only one exit arc from the full acyclic interval, the two forward paths from a decision node in the interval must subsequently merge into a single path, at a collector node. Thus, to each decision node in a full acyclic interval, there corresponds exactly one collector node.
Definition 2.7: A decision element is a subgraph consisting of a decision node and a collector node such that two forward paths, one from each side of the decision node, meet at the collector node.

A decision element may be denoted by $\{N_1, N_2\}$ where $N_1$ is the decision node and $N_2$ is the collector node at the junction of the two forward paths, one from each side of $N_1$.

There can be a subgraph on one or both of the forward paths from the two sides of $N_1$ to $N_2$. If the subgraphs on both of the forward paths from the two sides of $N_1$ to $N_2$ are smooth subgraphs, then, $\{N_1, N_2\}$ is an alternation; else, $\{N_1, N_2\}$ is not an alternation.

Six different types of decision elements are shown in chart IX; of these six, only chart IXa is an alternation. A dotted line in the chart represents any number of input or output arcs.

A decision element, $D_2$, is a descendant of another decision element, $D_1$, if a forward path from the decision node of $D_1$ to the collector node of $D_1$ passes through the decision node of $D_2$. Alternatively, $D_1$ is the ancestor of $D_2$.

In chart Xa, the decision element $\{1, 7\}$, type a, is a descendant of the decision element $\{11, 14\}$, type b. In chart Xb, the decision element $\{11, 16\}$, type c, is a descendant of each of the decision elements $\{11, 14\}$, type b, and $\{11, 16\}$, type c.
CHART IX
Decision Elements
CHART X

Two Examples of Decision Elements

(a)

(b)
Theorem 1.

A full acyclic interval is a smooth acyclic interval if and only if every decision element in the interval is an alternation.

Proof.

(i) If every decision element in a full acyclic interval is an alternation, then, the full acyclic interval is a smooth acyclic interval.

An alternation is a two-terminal subgraph which is generable by the graph grammar, SG, i.e., an alternation is a smooth subgraph.

Case I: Every decision element has no descendant. The full acyclic interval is then either a single decision element or a combination of computation nodes and decision elements. In either case, every two-terminal subgraph in the interval is generable by the graph grammar SG. Hence, the full acyclic interval is generable by the graph grammar SG, i.e., the full acyclic interval is a smooth acyclic interval.

Case II: Some decision elements have descendants. Consider the decision elements which have no descendants but have ancestors. By assumption, each such decision element is an alternation. Reduce the alternation to a single node. The previous ancestor of the alternation now becomes a decision element without a descendant. This decision element is also by assumption an alternation. Again, reduce it to a single node. The process is repeated until every decision element has no descendants. The interval is thus reducible to that in Case I above which has been proved to be a smooth acyclic interval.
(ii) if a full acyclic interval is a smooth acyclic interval, then, any decision element in the interval is an alternation.

If a full acyclic interval is smooth, then, the full acyclic interval is generable by the graph grammar SG. Only the productions P3.1, P3.2 and P3.3 (and P1.1, P1.2, P1.3 and P2) are capable of generating decision elements. But the decision elements generated by these productions are always alternations. Therefore, any decision element in the smooth interval is generable by the graph grammar, SG, and the decision element is always an alternation.

Q.E.D.
ALGORITHM A1.

To determine whether a given full acyclic interval is a smooth acyclic interval.

begin
if interval contains exactly one node
 then interval is smooth;
else begin
  identify each decision node and its ancestry level;
  if ancestry level of every decision node is zero,
    then, interval is smooth;
  else begin set NALT = 0;
    do while NALT = 0 and there is a decision node whose ancestry ≠ 0; identify decision elements which have no descendents;
      if any of these decision elements is not an alternation,
        then set NALT = 1
      else reduce each of the decision elements to a single node;
    end;
    if NALT = 1
      then, the interval is non-smooth;
    else the interval is smooth;
  end;
end;
2.3.1.2 Full Cyclic Intervals.

A full cyclic interval is a two-terminal interval which contains a cycle. The general form of a full cyclic interval is shown in Chart XI.

From definition 2.3, it is easy to show that the entry (or header) node, H, of a cyclic interval is always a collector node. T is the iteration node. Each of S1, S2 and S3 is an acyclic subgraph, otherwise, there would exist in the interval, one or more cycles which do not pass through the header node, H. But this would be a contradiction of definition 2.3. Therefore, each of S1, S2 and S3 is an acyclic subgraph.

The set of nodes in S1 is the pre-set. The set of nodes in S2 is the post-set. The set of nodes in S3 is the exit-set. The set of nodes consisting of H, S1, T, S2 is the loop-set. S3 may or may not be empty. Either S1 or S2 (but not both) may be empty.

Identification of the parts of a Cyclic Interval.
The parts of a cyclic interval are:
the header node,
the iteration node,
the loop-set,
the pre-set,
the post-set, and
the exit-set.

It is assumed that the header node, H, of the cyclic interval is known.
**Algorithm A2.**

Identification of the loop-set of a given cyclic interval.

```plaintext
begin
    initiate an expanding set, L, to be empty;
    initiate a set, P, to contain H, the header node;
    the elements of P are to be selected, one at a time, the
    next element to be selected being designated P^1;
    P^1 = H;
    \Sigma = \text{predecessors (P^1)};
    do while \Sigma \neq H;
       begin
          add \Sigma to P;
          remove P^1 from P and add it to L;
          P^1 = \text{next element of } P;
          \Sigma = \text{predecessors (P^1)};
       end;
    end;
end;
```

The iteration node is characterized by the following properties:

i. it is a decision node;

ii. it is in the loop-set;

iii. one of its two immediate successors is in the loop-set and
    the other is not.

The pre-set are those nodes in the loop-set which succeed the
header but precede the iteration node. The post-set are those
nodes in the loop-set which succeed the iteration node but precede
the header node.
The exit-set are those nodes in the interval which are not in the loop-set.

Theorem 2.

A full cyclic interval is a smooth cyclic interval if and only if each of its pre-set, post-set and exit-set is a smooth acyclic subgraph.

Proof.

i. If each of the pre-set, post-set and exit-set of a full cyclic interval is a smooth acyclic subgraph, then, the full cyclic interval is a smooth cyclic interval.

Suppose each of S1, S2 and S3 of Chart XI is a smooth acyclic subgraph, i.e., Chart XI is a full cyclic interval such that each of its pre-set, post-set and exit-set is a smooth acyclic subgraph.

Consider the subgraph generated by a sequence of productions of grammar SG as shown in Chart XII.

Each subgraph, S, in Chart XII is generable by the grammar SG, i.e., each subgraph, S, is a smooth subgraph. If each subgraph, S, in Chart XII is acyclic, then Chart XI is identical to the subgraph of Chart XIIe. But the subgraph of Chart XIIe is generated by grammar SG. Thus, Chart XI is generable by grammar SG, i.e., it is smooth.

ii. If a full cyclic interval is smooth, then, each of its pre-set, post-set and exit-set is a smooth acyclic subgraph.
CHART XI

General Form of a Smooth Cyclic Interval
CHART XII

Generation of a Cyclic Interval by SG

\[ S \rightarrow C \rightarrow l \rightarrow S \rightarrow S \rightarrow S \rightarrow l \rightarrow S \rightarrow S \rightarrow S \rightarrow S \rightarrow S \rightarrow S \rightarrow S \]
Suppose Chart XI is a smooth cyclic interval. Then, Chart XI is generable by the graph grammar SG. Therefore, each of S1, S2 and S3 is generable by SG, i.e., each of S1, S2 and S3 is smooth. But each of S1, S2 and S3 is acyclic, for, by definition, any cycle in an interval passes through the header node. The presence of a cycle in either S1 or S2 or S3 would contradict this definition. Therefore each of S1, S2 and S3 is acyclic.

Hence each of S1, S2 and S3 is both smooth and acyclic, i.e., each is a smooth acyclic subgraph.

Q.E.D.

It follows from theorem 2 above that if either the pre-set or the post-set or the exit-set is non-smooth, then the full cyclic interval is non-smooth. The pre-set or the post-set of the exit-set would be a non-smooth acyclic subgraph if:

(i) it is not a two-terminal subgraph or

(ii) a decision element in the subgraph is not an alternation.

If the post-set is not a two-terminal subgraph as shown in Chart XIII, then the exit from the loop at node S22 is an invalid loop-exit.

ALGORITHM A3.

To determine if a given full cyclic interval is a smooth cyclic interval.

begin

identify the loop-set, iteration node, pre-set, post-set and exit-set;

use algorithm A1 to check if any of the pre-set, post-set
CHART XIII

Invalid Loop Exit
or exit-set is non-smooth;

if any of the subgraphs is non-smooth,
then, the full cyclic interval is non-smooth;
else, the full cyclic interval is a smooth cyclic interval
end

2.3.2. Deferred Interval Reduction of a Flowgraph.

Algorithm A4, given below is a "bottom-up" parsing algorithm. It looks for two-terminal intervals, checks if each two-terminal interval is generable by the graph grammar, SG, and if so, reduces each of such intervals to a single node, thus obtaining a smaller, higher-order flowgraph. If possible, the process is repeated until the flowgraph is reduced to a single node, in which case the flowgraph is deferred reducible.

If at any point in the process, a two-terminal interval which is not generable by the graph grammar, SG is identified, or a higher-order flowgraph is obtained such that it contains more than one node but cannot be reduced any further, then, the given flowgraph is not deferred reducible.

There is a difference between deferred interval reduction and Cocke-Allen's interval reduction (9), (10). In Cocke-Allen's interval reduction, any interval is reducible. In deferred interval reduction, only two-terminal, smooth intervals are reducible.
ALGORITHM A4. Deferred Interval Reduction.

begin set IRDCBL = 0
    do until current flowgraph is a single node or IRDCBL = 1;
        begin partition current flowgraph into intervals;
            identify full intervals (if any);
        do while IRDCBL = 0
            for each full interval which is not a single node;
                if interval is acyclic
                    then use algorithm A1;
                else use algorithm A3, to determine whether
                    the full interval is smooth;
                if interval is non-smooth
                    then set IRDCBL = 1;
                else reduce the full interval to a single node;
            end;
        if IRDCBL = 0 and current flowgraph contains the same
            number of nodes as the immediate low-order flowgraph
            then set IRDCBL = 1;
    end;
    end;
if IRDCBL = 1
    then the given flowgraph is not deferred reducible;
else the given flowgraph is deferred reducible;
end
Examples of Deferred Interval Reduction.

Apply algorithm A4 to the flowgraph shown in Chart XIVa. The intervals of the flowgraph are:

$I(1) = \{1,2,3\}$
$I(4) = \{4,5,6,7\}$
$I(8) = \{8,9\}$

Interval $I(1)$ is a partial interval; so also is interval $I(8)$. These intervals are left unchanged.

Interval $I(4)$ is a full cyclic interval. It has a single valid exit. It is a smooth cyclic interval. It may therefore be reduced to a single node.

The first-order flowgraph is shown in Chart XIVb. Again, apply algorithm A4 to the flowgraph of Chart XIVb. The interval of this flowgraph is $I(1) = \{1,2,3,10001,8,9\}$.

$I(1)$ is a full acyclic interval. It contains only one decision element and this is an alternation. $I(1)$ is a smooth acyclic interval. It is reduced to a single node.

The second-order flowgraph is shown in Chart XIVc. This flowgraph consists of a single node. Therefore, the given flowgraph of Chart XIVa is deferred reducible; hence, it is smooth.

Apply algorithm A4 to the flowgraph shown in Chart XV. The intervals of that flowgraph are:

$I(1) = \{1,2,4\}$
$I(3) = \{3,5,6,7,8,9,10,11,12,13,14\}$
$I(15) = \{15,16\}$
CHART XIV

Deferred Interval Reduction

Diagram a

Diagram b

Diagram c
CHART XV

A Non-Smooth Flowgraph
Intervals I (1) and I (15) are partial intervals. They are left unchanged.

Interval I (3) is a full cyclic interval. It has a single valid exit. However, the decision element \{9,14\} within the interval is not an alternation because it has three terminals at 9,10 and 14. The interval cannot be reduced into a single node without first transforming the decision element \{9,14\} into an alternation. The flowgraph is non-smooth.

Theorem 3.

A flowgraph is a smooth flowgraph if and only if it is deferred reducible.

Proof.

1. if a flowgraph is deferred reducible, then it is a smooth flowgraph.

Consider a graph which is known to be deferred reducible. Then, it is possible to identify two-terminal subgraphs of the flowgraph and corresponding productions of SG by which each two-terminal subgraph can be reduced to a single node. Such reductions can be repeated until the flowgraph is reduced to a single node. It is then possible to perform the opposite procedure, i.e., beginning with the start symbol S, apply appropriate productions of the grammar SG, to obtain a flowgraph identical to the given flowgraph. Thus, the given deferred reducible flowgraph is generable by the graph grammar SG; i.e., it is a smooth flowgraph.

ii. if a flowgraph is smooth, then, it is deferred reducible.
Consider a smooth flowgraph. Since it is smooth, it is generable by the graph grammar, SG. This implies that every two-terminal subgraph of the given flowgraph is generable by SG. Thus, it is possible to begin with the start symbol S, apply appropriate productions of SG and obtain the given flowgraph. Then, it is also possible to perform the same operations in reverse, i.e., identify the "innermost" two-terminal subgraphs; since each of these is generable by SG, there is a production of SG by which it can be reduced to a single node. Let it be reduced to a single node. The former flowgraph is thus reduced to a smaller flowgraph. However, the two-terminal subgraphs of the smaller flowgraph are a subset of the two-terminal subgraphs of the original flowgraph. Therefore, the two-terminal subgraphs of the new flowgraph are generable by SG. Thus the innermost two-terminal subgraphs of the new flowgraph can be reduced to single nodes to obtain a new still smaller flowgraph. It follows that any new flowgraph obtained by reduction of the original smooth flowgraph contains two-terminal subgraphs which are a subset of a subset of ... of a subset of the two-terminal subgraphs of the original flowgraph and each of these is generable by SG. Thus, it is always possible to continue the reduction procedure until a single-node flowgraph is obtained; i.e., the original flowgraph is deferred reducible. Q.E.D.
2.4 Transformation of a non-smooth interval to a smooth interval.

The corollary to theorem 1 is that a full acyclic interval is non-smooth if there exists in the interval a decision element which is not an alternation. Examples of decision elements which are not alternations are shown in Chart IXb and IXc.

It can be deduced from theorem 2 that a full cyclic interval is non-smooth if one or both of the following conditions are satisfied:

(i) there exists in the interval a decision element which is not an alternation.

(ii) there exists in the interval an invalid loop-exit.

An invalid loop-exit is shown in Chart XIII. The conclusion can then be drawn that a full interval is non-smooth if either there exists in the interval a decision element which is not an alternation or there exists in the interval an invalid loop exit or both of above conditions are present.

A non-smooth interval can then be transformed into a smooth interval by correcting each of these two conditions whenever it arises.

2.4.1 Transformations on decision elements which are not alternations.

A decision element is not an alternation if one (or both) of the subgraphs on the two paths from its decision node to its collector node is not a two-terminal subgraph. The decision element can be transformed into an alternation by making each of these subgraphs a two-terminal subgraph, but preserving the original logic of the interval. In the case of a decision element which has no descendants, the transformation can be accomplished by path splitting.
In Chart XVIa, the decision element \( \{3,6\} \) is not an alternation because the subgraph on the path 3-7-4-5-6 is not a two-terminal subgraph. Notice that \( \{3,6\} \) has no descendant and that \( \{3,6\} \) contains two collector nodes.

It is true, in general, that a decision element which has no descendant and is not an alternation contains multiple collector nodes. The collector node which is at the junction of the two paths from the decision node of the decision element is the valid collector node of the decision element. The other collector nodes are invalid collector nodes.

Path splitting consists of copying the path from each invalid collector node to the valid collector node of the decision element. Each invalid collector node is moved from its previous position to the end of the copied path. In Chart XVIa, node 6 is the valid collector node of the decision element \( \{3,6\} \). Node 4 is an invalid collector node in \( \{3,6\} \). \( \{3,6\} \) is transformed into an alternation by splitting the path from invalid collector node 4 to the valid collector node 6 as in Chart XVIb. Note that the former invalid collector node 4 has now been moved to a valid position at the end of the split path.

The path splitting procedure outlined above transforms decision elements which are not alternations and which have no descendants. What of decision elements (which are not alternations) which have descendants? A reduction process, outlined below, is applied until such decision elements have no descendants and then, the previous procedure would be applicable. The reduction process is as follows:
CHART XVI

Path Splitting

a

b
Identify the decision elements which have no descendants. Transform, by path-splitting, those of them which are not alternations into alternations. Reduce each of the decision elements (which are now all alternations) to a single node. Some decision elements which previously were ancestors now do not have descendants. Repeat the process until all decision elements in the given interval have been reduced. The corrected interval can now be obtained by expanding each reduced element to its full form.

ALGORITHM A5

Transformation of a full, non-smooth, acyclic interval to a smooth acyclic interval.

begin do while there are decision elements;
        identify decision elements which have no descendants;
        do for each such decision element;
                 if decision element is not an alternation,
                          then use path-splitting to transform into an alternation;
                 reduce each alternation to a single node;
                 if there is a split path,
                          then construct a path to join the collector node at the end of the split path to the appropriate (collector) node in the interval;
        end;
end;
recover the transformed interval by expanding each reduced element to its full form;

end

Apply algorithm A5 to the interval shown in Chart XVII. \{7, 12\} and \{17, 19\} are decision elements without descendants. \{17, 19\} is an alternation but \{7, 12\} is not an alternation. Node 8 is an invalid collector node in \{7, 12\}. \{7, 12\} is transformed into an alternation by splitting the path 8-11-12. The new alternation \{7, 12\} and the alternation \{17, 19\} are reduced to single nodes to obtain Chart XVIIIa. Apply algorithm A5 to Chart XVIIIa. \{4, 8\} and \{3, 21\} are decision elements without descendants. \{4, 8\} is an alternation but \{3, 21\} is not. \{3, 21\} is transformed into an alternation by splitting the path 14-17-19-21. The new alternation \{3, 21\} and the alternation \{4, 8\} are reduced to single nodes to obtain Chart XVIIIb.

The reduction and transformation processes are continued until a single node is obtained.

The transformed, smooth interval is recovered by expanding each reduced node into the alternation from which it was derived. The recovery process is illustrated in Chart XIX.

2.4.2 Transformations on invalid loop-exits.

A cyclic interval contains a loop. The valid exit from the loop is at the iteration node. An exit from the loop at any other node is an invalid loop-exit. Chart XIII illustrates an invalid loop-exit. A cyclic interval contains an invalid loop-exit if its post-set (or exit-set) is not a two-terminal subgraph.
CHART XVII

A Non-Smooth Acyclic Interval
CHART XVIII

Transformation of a Full Non-Smooth Acyclic Interval
CHART XIX

Recovery of Transformed Flowgraph

a

b

c
CHART XIX

Recovery of Transformed Flowgraph
CHART XIX

Recovery of Transformed Flowgraph
An invalid loop-exit can be eliminated by transforming the post-set (or exit-set) into a two-terminal subgraph, while preserving the logic of the interval and such that the only exit from the loop is from the iteration node. This can be achieved by introducing a new boolean variable for each invalid loop-exit. Each such boolean variable is set to "false" prior to entry to the loop and it is set to "true" at the invalid exit from the loop. To preserve the original execution sequence, it is necessary to test the new boolean variables, prior to each repetition of the loop, so that a valid exit, replacing the invalid ones, can be made. A new decision node at which the test of the boolean variables is effected becomes the iteration node, thus displacing the original iteration node. This displacement has the effect of creating an "artificial" invalid loop-exit at the original iteration node. This effect is neutralized by introducing another boolean variable at the "artificial" invalid loop-exit.

Chart XX illustrates a full cyclic interval with an invalid loop-exit at node 6.

Note that each of the post-set and exit-set of Chart XX is not a two-terminal subgraph. To eliminate the invalid loop exit at node 6, a boolean variable, B1, is set to "true" after node 6, at node 13, of Chart XXI. Because boolean variables must be tested prior to repetition of the loop, the original iteration node 3 has been displaced by the new iteration node 11. Because of this displacement,
CHART XX

Subsets of the Nodes of a Full Cyclic Interval
CHART XXI

Correcting for Invalid Loop-Exits

B1 = 'false'
B2 = 'false'

17

1

B1 v B2

2

3

4

6

7

8

9

10

11

12

13

14

15

16

5

615
A boolean variable, B2, is introduced after node 3 (B2 is set to "true" at node 12) to eliminate the "artificial" invalid loop-exit created at node 3. The boolean variables, B1 and B2, are set to "false" (at node 17) prior to entry to the loop.

**Algorithm A6.**

Transformation of a full, non-smooth, cyclic interval to a smooth cyclic interval.

Note that a full cyclic interval has a single entry to the loop (or cycle) in the interval through the interval header.

**begin**

Determine the loop-set and the iteration node as defined in section 2.3.1.2.

if there is an invalid loop-exit

then begin

    do for each of all exits;

    note the "target node" of the exit; sever the exit arc; create a new node at the severed point; set a new boolean variable to "true" at the new node;

    join the new node to the loop-header;

end;

create a new node, P, as the immediate predecessor of the loop header; set all the newly introduced boolean variables (n in number, say) to "false" at P; denote by S, the immediate successor of the loop header; create a new iteration node
between the loop-header and S; the predicate at the new
iteration node is the logical conjunction of all the introduced
boolean variables; the "false" arm of the test is incident on
S; the "true" arm of the test is incident on a two-terminal,
n-way switching subgraph by which each boolean variable is
used to reach its corresponding "target node";

end;

if the interval contains decision elements which are not alternations
then for each decision element which is not an alternation

    do:
        call algorithm A5 to transform into an alternation;
    end;

end;

2.4.3. Transformations on other irreducible flowgraphs.

In section 2.3.2, it was pointed out that a flowgraph is not
defered reducible if either:

i. it contains a non-smooth interval or

ii. its highest-order reduced flowgraph contains more than
    one interval.

The transformations discussed in section 2.4.1 and section
2.4.2 are applicable to flowgraphs of the first type above, i.e.,
full intervals which are non-smooth.

This section deals with flowgraphs of the second type above,
i.e., flowgraphs which are not deferred reducible because their
highest order flowgraphs contain more than one interval.
Some or none of the intervals in such a flowgraph can be full. However, any full interval in such a flowgraph consists of exactly one node. Since deferred reduction causes only full intervals to be reduced to single nodes, no further reduction of the flowgraph can be achieved by reducing its full, single-node intervals.

Two examples of highest-order flowgraphs which are not deferred reducible are shown in Chart XXII.

Theorem 4.

If a highest-order flowgraph is not a single node, then, it has a minimum of three intervals.

Proof.

Since the flowgraph cannot be deferred reduced to a single node, it contains more than one interval.

Suppose the flowgraph contains exactly two intervals. Note that the flowgraph has a single flowgraph entry and a single flowgraph exit.

Case I: The first interval, containing the flowgraph entry is a full interval.

Since it is a full interval, the first interval has a single exit. Since there are only two intervals in the flowgraph, the exit arc from the first interval is also the entry arc to the second interval. Therefore, the second interval has a single entry. Also, the second interval has a single exit, the flowgraph exit. Therefore, the second interval has a single entry and a single exit; i.e., it is a full interval. Hence, each of the two intervals is a full interval. But if this
CHART XXII

Two Examples of Highest-Order Flowgraphs which are not Deferred Reducible

Diagram a: 
Diagram b:
were so, the flowgraph would be deferred reducible. Therefore, if the first interval is a full interval then, there are more than two intervals in the flowgraph.

Case II: The first interval has more than one exit. The exit arcs from the first interval are incident on different nodes. Each of these nodes is the header of an interval. Hence, if the first interval has multiple exits, then there is more than one interval in addition to the first, i.e., there are at least three intervals in the flowgraph.

Irrespective of the number of exits from the first interval, there are a minimum of three intervals in the flowgraph.

Q.E.D.

The theorem above can be verified as follows: Recall that when any given flowgraph is partitioned into intervals, the header of every interval, with the possible exception of the header containing the flowgraph entry, is a collector node.

Also, recall that in a highest-order flowgraph, no interval contains a cycle since the cyclic portion of any interval would have been reduced to a single node at the earlier stages of reduction.

If the highest-order flowgraph contains n intervals (n > 1), at least, n-1 of the interval headers are collector nodes. The two arcs incident on each of these collector nodes are "external" arcs in the sense that they originate from outside of the interval of which the collector node is the header. Hence, there are 2(n-1) "external"
arcs incident on the \( n-1 \) headers. However, there are only \( n \) distinct intervals.

If \( 2(n-1) \geq n \), then,

\[
\begin{align*}
n-2 &> 0; \quad n \geq 2 \\
\text{minimum (n)} &= 3.
\end{align*}
\]

**Theorem 5.**

If a highest-order flowgraph is not reducible to a single node, then, the flowgraph contains a cyclic subgraph with multiple loop entries.

**Proof.**

It is shown above that in a highest-order flowgraph which is not reducible to a single node, the number of "external" arcs incident on its interval headers is greater than the number of distinct intervals. There exist at least two of the intervals such that an exit arc of one is an "external" arc of the other. A set of such intervals form a subgraph containing a loop. The entries to this loop are through those "external" arcs which are not part of the subgraph.

Suppose a set of \( m \) (\( m \geq 1 \)) intervals form a subgraph containing a loop. At least \( m \) of the "external" arcs to the headers form part of the subgraph. Therefore, there are a maximum of \( m \) "external" arcs incident on the headers from outside of the subgraph. However, it is possible that a subset, \( n \), of these \( m \) intervals have both "external" arcs within the subgraph. Since the "external" arcs of the header of an interval originate from two different intervals, then the
maximum possible value of \( n = (m-1)/2 \). Therefore, the minimum value, \( e \), of truly external arcs incident on the subgraph is

\[
e = m - n \\
= m - (m-1)/2 \\
= (m+1)/2.
\]

Since \( m > 1 \), then, \( e > 1 \); i.e., there are multiple entries to the loop.

Q.E.D.

Two examples of highest-order flowgraphs which are not deferred reducible are shown in Chart XXII. They illustrate the points made above. The intervals of Chart XXIIa are:

\[
I(1) = \{1, 2, 3\} \\
I(4) = \{4, 5\} \\
I(6) = \{6, 7, 8\}
\]

Two of these intervals, \( I(4) \) and \( I(6) \) form a subgraph containing a loop. Two of the "external" arcs \( (a_2, a_3) \) incident on the interval headers form part of the subgraph containing the loop. Entries to the loop are through the "external" arcs \( a_1 \) and \( a_4 \).

The intervals of Chart XXIIb are:

\[
I(1) = \{1\} \\
I(2) = \{2, 3, 5\} \\
I(4) = \{4, 6, 7\} \\
I(8) = \{8, 9, 10\}
\]

Three of these intervals, \( I(2) \), \( I(4) \) and \( I(8) \) form a subgraph containing a loop. One of the three headers to these intervals (header 4) has both "external" arcs \( a_3 \) and \( a_4 \) within the
subgraph. Each of the two other headers has only one "external" arc within the subgraph. The entries to the loop from outside of the subgraph are through the "external" arcs $a_1$ and $a_6$.

A loop generable by the graph grammar, SG, has a unique entry; therefore, a flowgraph containing a loop with multiple entries is not generable by SG.

2.4.3.1. Correction for Multiple Loop-entries.

A preferred entry is chosen among the multiple loop entries. Each of the other entries is effectively removed by splitting the path from the unwanted entry to the preferred entry. The choice of the preferred entry is dependent on the iteration node of the loop and the choice of the iteration node depends on the exits from the loop. If there is only one exit from the loop, then the node at which the exit occurs is the iteration node. In that case, the preferred entry is that which is the closest predecessor of the iteration node. If there are multiple exits from the loop, a correction should first be made to remove all the exits but one. The procedure required is similar to that outlined in section 2.4.2. After the invalid loop-exits have been removed, the resulting flowgraph might contain full intervals which are reducible. Interval analysis should then be performed to determine if this is so.

Algorithm A7. Transformations on Multiple Loop-entries.

\begin{verbatim}
begin
  identify a maximal, strongly connected region of the flowgraph;
  ALTER = '0'B;
\end{verbatim}
do while (7 ALTER);
    identify the next loop in the region;
    if multiple entries to the loop
    then do; ALTER = 'l'B;
        determine number of exits from loop;
        if multiple exits
            then do;
                correct for invalid loop-exits;
            end;
        else do;
            determine the preferred entry;
            apply path splitting to remove other entries;
        end;
    end;
end;

perform deferred interval reduction on the transformed flowgraph;
end;

Apply algorithm A7 to Chart XXIIa.

Multiple entries to the loop are found at 4 and 6. There is a single exit from the loop at node 7. Hence, the preferred entry is at node 6. The unwanted entry at 4 is removed by splitting the path from 4 to 6. The resulting flowgraph is shown in Chart XXIIIa. This flowgraph contains reducible full intervals.

Apply algorithm A7 to Chart XXIIb.

The maximal strongly connected region is \( \{2, 6, 4, 9, 8, 5, 3\} \).

Multiple entries to the region occur at 6 and 9. Corrections
performed to remove multiple exits result in the flowgraph shown in Chart XXIIIb. This flowgraph contains a full interval which can be reduced and another interval which contains a loop with two entries. One of these loop-entries can be removed after further transformations.
CHART XXIII

Corrections for Multiple Loop-Entries

a

b
Consider any flowgraph, specified in the form \((V,A)\), where \(V\) is the set of nodes of the flowgraph and \(A\) is the set of arcs of the flowgraph.

Let a structural property, \(S\), of the flowgraph be defined as a function of \(V\) and \(A\), i.e.,
\[
S = f(V,A) \tag{3.1}
\]

The index of smoothness, \(\sigma\), is defined as a function of \(S\), i.e.,
\[
\sigma = g(S) \tag{3.2}
\]

It is desirable to fix a limited range within which the numerical value of the index of smoothness, \(\sigma\), may lie. The range \((0,1)\) is considered suitable. If a flowgraph is smooth, it is assigned an index of 1; otherwise, it is assigned an index whose value is less than 1 depending on the function \(g\) in equation 3.2.

Now, what are these functions, \(f\) and \(g\) in equations 3.1 and 3.2?

\(g\) is a function of the structure of the flowgraph. Algorithms A1 and A3 discussed in Chapter 2 can be used to determine whether a given flowgraph is or is not smooth. If the flowgraph is found to be non-smooth, then, one of algorithms A5, A6 or A7 can be used to transform the flowgraph to a smooth one.

Consider a given flowgraph, \((V,A)\). Denote its structure, \(S_1\) by
\[
S_1 = f(V,A) \tag{3.3}
\]
Determine if the flowgraph is smooth. If not, then, transform the flowgraph to an equivalent smooth flowgraph. If transformations are performed, the resulting flowgraph has a structure different from that of the given flowgraph, i.e., \( V \) and \( A \) may now have new values \( V' \), \( A' \) respectively. The new structure may be denoted by

\[ S_2 = f (V', A') \] \(-\ldots-3.4\)

The function \( g \) may be regarded as a measure of the relative change in the structure of the flowgraph in transforming it into a smooth flowgraph, i.e.

\[ \sigma = g(S) = \frac{S_1}{S_2} \] \(-\ldots-3.5\)

Note that equation 3.5 can be put in the form:

\[ \sigma = \frac{S_1}{S_1 + T} \] \(-\ldots-3.6\)

where \( T \) represents the changes to the structure of the flowgraph during necessary transformations. If the given flowgraph is smooth, then, no transformations are necessary, i.e., \( T = 0 \) and then, \( \sigma = S_1/S_1 = 1 \). If transformations are necessary, then, \( T > 0 \) and \( \sigma < 1 \). Assuming that any given flowgraph has a positive, non-zero value of \( S \), then, \( \sigma > 0 \). Thus,

\[ 0 < \sigma \leq 1 \] \(-\ldots-3.7\)

The function \( f \) can be determined by a consideration of the transformations performed on non-smooth flowgraphs.

Algorithm A5 is used to transform a full, non-smooth acyclic interval into a smooth one. This requires splitting of paths between certain pairs of nodes in the flowgraph.
Algorithm A6 is used to transform a full, non-smooth cyclic interval into a smooth cyclic interval. This requires a rearrangement of some of the nodes and the use of boolean variables to correctly reconnect those nodes.

Algorithm A7 is used to remove multiple entries into loops. This requires path splitting and in some cases, boolean variables are used too.

From the nature of these transformations, it may be surmised that the function \( f \) is related to the paths in the flowgraph and to the boolean variables used in the transformation processes. However, a common effect is produced on a flowgraph by path-splitting and by use of boolean variables with rearrangement of some nodes. That common effect is an increase in the lengths of some of the paths through the flowgraph.

Thus, the function \( f \) can be identified with the total number of arcs of the flowgraph, i.e.,

\[
S = \sum_i |\Gamma(V_i)| \quad \text{---3.8}
\]

where \( \Gamma \) is the successor function and \( |\Gamma(V_i)| \) is the cardinality of the immediate successors of node \( V_i \).

By combining equations 3.3, 3.4, 3.5 and 3.8, the index of smoothness,

\[
\sigma = \frac{\sum_i |\Gamma(V_i)|}{\sum_j |\Gamma(V_j)|} \quad \text{---3.9}
\]

Consider the Knuth-Floyd (3) example program:

\[
\text{begin} \\
\text{for } i: = 1 \text{ step 1 until } n \text{ do} \\
\text{if } A(i) = \times \text{ then go to found;}
\]

(3) (4)
The flowgraph of the program is shown in Chart XXIVa. The number in brackets at the end of each line of the program is the identification number of the node of the flowgraph which corresponds to that line of program statements.

The flowgraph of Chart XXIV is non-smooth. After the application of algorithm A6 to chart XXIVa, the smooth flowgraph of Chart XXIVb is obtained. A translation of the smooth flowgraph into a source language program is as follows:

```
begin
  located: = false; exhausted: = false; i: = 1;
  while not located and not exhausted do;
    begin
      if i > n
        then exhausted: = true
        else if A(i) = x
          then located: = true
          else i: = i + 1
    end;
  if not located
    then begin n: = i; A(i): = x; B(i): = 0 end;
  B(i): = B(i) + 1
end
```
CHART XXIV

Calculation of Index of Smoothness

Diagram a

Diagram b
The cardinality of the arcs of Chart XXIVA,

\[ S_1 = \sum_{i=1}^{8} |\Gamma(V_i)| \]

\[ = (1+1+2+2+1+1+1+0) \]

\[ = 9 \]

The cardinality of the arcs of Chart XXIVB,

\[ S_2 = \sum_{j=1}^{15} |\Gamma(V_j)| \]

\[ = (1+1+1+2+2+1+1+1+1+2+1+1+1+1+0) \]

\[ = 18 \]

\[ = g(S) \]

\[ = S_1/S_2 \]

\[ = 0.500 \]

The index of smoothness of Chart XXIVA is 0.500.
CHAPTER 4

EXPERIMENTAL DATA

4.1. Data Base.

In order to assess the usefulness of the index of smoothness, it is necessary to determine its values for a substantial number of flowgraphs. The relation of the index to other variables of the flowgraph can then be analyzed.

The programs published in the "Algorithms" section of the Communications of the Association for Computing Machinery (CACM) is considered a suitable data base.

Seventy-eight of these algorithms were randomly selected (15). The flowgraph of each algorithm was constructed manually. It was then coded and supplied as input data to program "SMOOTH" which is briefly described in the next section. An output of program "SMOOTH" is the index of smoothness of the input flowgraph.

4.2. Program "SMOOTH"

Program "SMOOTH" is a combination of algorithms A1 to A7 (discussed in Chapter 2) into a unified working whole. The flowgraph of the program is shown in Chart XXV. Brief descriptions of some features of the program are given in this section. A summary of the program is given in the Appendix.

Program SMOOTH has the capability to analyze either a single flowgraph or a batch of flowgraphs in a sequence.
CHART XXV

Flowgraph of Program "SMOOTH"

START

Accept, Store & Display Flowgraph

Determine the Intervals

Only one single-node interval?

Yes

XXX C

No

XXX B
CHART XXV

Flowgraph of Program "SMOOTH"

Yes

Correct for multiple loop-entries

No

Any non-smooth full intervals?

No

Transform non-smooth intervals to smooth

Yes

Reduce each smooth interval to a single node
Any previous transformations?

Yes:
- Calculate index of smoothness
  - Recover & Display smoothed flowgraph

No:
- Index of smoothness = 1

STOP
Some tracing capabilities are also provided as options which a user can exercise. These trace options enable the user to display, as a printed output, the process or the result of one or more aspect of the flowgraph analysis. The trace options provided are as follows:

1: display the process of partitioning the flowgraph into intervals. This option should be exercised with care since it may cause the production of large printed output.

2: display the intervals of the flowgraph.

3: display every loop and loop-exit.

4: identify every decision element which is not an iteration.

5: display every loop which has multiple entries.

6: identify all reduced intervals in the final flowgraph.

7 & 8: reserved.

For batch analysis of flowgraphs, the options are global, i.e., the options apply to the current flowgraph and all subsequent flowgraphs in the batch. However, changes to the global options can be specified locally for any flowgraph.

The non-optional outputs of the program are:

i. a printed output of the given flowgraph.

ii. the index of smoothness of the given flowgraph and if the given flowgraph is non-smooth, then,

iii. a printed output of the transformed flowgraph.

4.2.1. Data Structure.

Before a flowgraph can be analyzed by the program "SMOOTH," the flowgraph has to be coded as data and supplied to the program. A
flowgraph can be represented inside a computer in one of several ways: either as a matrix, or as a tree, or as a linked-list, etc. The two main considerations that influenced the choice of the data structure for this study are:

i. the program should be able to handle large flowgraphs, containing hundreds, possibly thousands, of nodes.

ii. the transformations on non-smooth flowgraphs, as discussed in Chapter 2, should be done efficiently.

From these considerations, a multi-linked structure was considered the best choice.

The unit of this structure is the node. Each node is represented, as a PL/I structure as follows:

1 Node,
   2 NIDENT Fixed Binary (15),
   2 ITVHDR Pointer,
   2 ITVTLR Pointer,
   2 NABONODE (4) Pointer;

NIDENT is the identification number of the node. ITVHDR and ITVTLR are usually NULL pointers unless the node represents a reduced interval; then, ITVHDR and ITVTLR point to the header node and the tail-node of the reduced interval. These two pointers play an important part in the deferred interval reduction discussed in Chapter 2. NABONODE is a four-element vector, pointing to the "neighbors" of the node. NABONODE (1) is the first (or left) predecessor, NABONODE (2) is the second (or right) predecessor, NABONODE (3) is the first (or left) successor and NABONODE (4) is the second (or right) successor.
The data structure above is in practical agreement with the definition, in Section 2.1.2, of a standard flowgraph. A node of a standard flowgraph has a maximum of two predecessors, a maximum of two successors and a maximum of three "neighbors."

4.2.2 Coding of a Flowgraph for Program Input

It is assumed that the flowgraph to be analyzed is already constructed. The nodes of the flowgraph are assigned identification numbers in a convenient manner. These identification numbers need not be in ascending or descending order. However, the identification assigned to any node must be a positive, non-zero number, not exceeding 9,999. None of the nodes of the initial flowgraph represents a reduced interval; hence ITVHDR, ITVTLR = NULL for every node of the initial flowgraph. This information is assumed and need not be supplied for each node by the coder. Thus, the coded data for the input of each node is of the form:

NIDENT, N1, N2, N3, N4

where N1 and N2 are the identification numbers of the predecessors and N3 and N4 are the identification numbers of the successors of the node. This "external" coded form is subsequently converted by the program into the "internal" multi-linked structure representation of the flowgraph.
As an example, the flowgraph shown in Chart XXIVa would be coded as follows:

```
1, 0, 0, 2, 0
2, 1, 4, 3, 0
3, 2, 0, 4, 5
4, 3, 0, 2, 6
5, 3, 0, 6, 0
6, 4, 5, 7, 0
7, 6, 0, 8, 0
8, 7, 0, 0, 0
```

4.2.3. Data for Production Run of SMOOTH.

The data required for the operation of the program are usually supplied on cards as follows:

i. NRVNS and GOPTIONS: FORMAT (F(5), X(4), F(1)). NRVNS is the number of flowgraphs in the batch. This number must be right-justified in columns 1 - 5. A non-zero value of GOPTIONS in column 10 indicates that trace options are exercised.

ii. TRACE-OPTIONS: FORMAT (8 B(1)).

This card should be included only if GOPTIONS above is non-zero. The options are specified in columns 1 - 8 as a string of 8 binary digits. Each of these digits must be either 0 or 1. No other character is allowed; specifically, a blank cannot be used instead of a zero. Each column represents the option whose number is the same as that of the column.
iii. KNODES, LOPTION and FLONAME: FORMAT (F(5), X(4), F(1), X(10), A (20)). KNODES is the total number of nodes in the flowgraph. The number must be right-justified in columns 1 - 5. A non-zero value of LOPTIONS in column 10 indicates that changes to the global options will be made. FLONAME is a 20-character name of the flowgraph.

iv. TRACE-OPTIONS: FORMAT (8 B(1)).
This card is required only if LOPTIONS in iii above is non-zero. The eight binary digits are punched in columns 1 - 8. These options replace any previous global options.

v. NODES: FORMAT (5(F(5), X (5)).
Each node of the flowgraph is coded and punched into a card as explained in 4.2.2. The number of cards in this data group is KNODES, specified in iii above.

For batch analysis of flowgraphs, iii, iv and v above are supplied for each flowgraph.

4.3. Some Other Variables of an Algorithm.

The other variables of an algorithm which may be related to its flowgraph are described in this section.

4.3.1. Explicit Transfers of Control (ETC)

The number of statements in the algorithms which cause explicit transfer of control. Examples of such statements are: switch, arithmetic if, goto, computed goto, return.
4.3.2. Critical Remarks (CR)

Some of the algorithms published in CACM have been tested and certified. In some of these cases, the certifiers have made some critical remarks, such as:

i. "the algorithm is too slow" or "efficiency can be improved"

ii. "storage efficiency can be improved"

iii. "there are better criteria for loop termination"

iv. "the algorithm can become unstable under certain conditions"

v. "Higher precision is required" or "round-off errors can be large"

The variable, CR, is the number of such remarks.

4.3.3. Errors (ERR)

Some certifiers point out errors in the algorithm. ERR is the number of these errors - excluding typographical errors.

4.3.4. Programming Language (LANG)

Most of the algorithms chosen for this study are written in ALGOL. The rest are written in FORTRAN. The indices of smoothness of the flowgraphs in the two groups can be used to test whether there is or there is not any significant difference between the two groups.

4.3.5. Time (TIME)

The year of publication of each algorithm is known. It can be determined whether there is any significant change with the passage of time in the index of smoothness of the "average" algorithm.

4.3.6. Geographical Region (REG)

The algorithms can be grouped according to their regions of origin and indices of smoothness on regional bases can then be
computed. As of 1974, there are 12 regions of ACM membership. The regions are shown in Table 14.

4.3.7. Logical Size (LS)

This is the number of nodes in the minimal flowgraph of the algorithm.

4.3.8. Density of Explicit Transfers of Control (DETC)

This is the ratio of the number of explicit transfers of control (ETC) to the number of segments (SEG) in the flowgraph.

4.3.9. Segments (SEG) (12)

A segment is defined as either a maximal loop or a maximal alternation or a computation block not contained in a loop or alternation. SEG is the number of segments in the minimal flowgraph of the algorithm.

4.3.10. Biggest Segment (BSEG) (12)

This is the number of nodes in the segment with the highest number of nodes.

4.3.11. Level of Nesting (NEST) (12)

A loop or an alternation can be nested in another loop or alternation. NEST is the level of nesting of the "innermost" loop or "innermost" alternation.

4.3.12. (DIFF)

This is the product of BSEG and NEST. It may be regarded as a rough measure of the difficulty of comprehension of the algorithm.
4.3.13. Average Logical Size of a Segment (ALS)

This is the ratio of the logical size (LS) to the number of segments (SEG).

4.3.14. Certifications (CERT)

This is the number of published certifications of the algorithm.

4.4 Analysis of Flowgraphs Data.

For the selected algorithms, the index of smoothness, $\sigma$, and corresponding values of the other variables are shown in Tables 1-8.

The SPSS (13) program was used for most of the statistical analyses.

4.4.1. Correlation of Index of Smoothness with Other Variables.

On an inspection of Tables 1-8, it is evident that the Probability Distribution Function (PDF) of the index of smoothness is not "normal" i.e., it is not "bell-shaped." The PDF's of SEG, BSEG and NEST may be regarded as approximately "normal," though with considerable skew. The PDF's of the other variables are not "normal." Considering that most of the PDF's are not normal, non-parametric statistical analysis is considered most appropriate for these data.

There are several "ties" in ranking the index of smoothness. Spearman's correlation is less sensitive to "ties" than Kendall's correlation; hence the correlation coefficients shown in Table 10, Table 11, and Table 12 are Spearman's correlation coefficients. Kendall's correlation coefficients for the data are generally slightly smaller. Pearson's correlation coefficients for the data in Tables 1-8 are generally smaller than Kendall's. However, the correlation coefficients of
either Spearman or Kendall or Pearson, for the data of Tables 1 - 8 consistently indicate the same trend.

Table 10 shows the correlation of the index of smoothness with the other variables for the total flowgraph population.

There is a negative and significant (at the 0.1% level) correlation between the index of smoothness and each of the following variables: ETC, LS, DETC, BSEG, NEST, DIFF and ALS. This means that the index of smoothness tends to decrease as each of these variables increase.

The correlation between the index of smoothness and each of CR and ERR is negative but not significant; however, the hypothesis that the index of smoothness decreases as either CR or ERR increases could be true, though, for CR, the hypothesis is false about 15% of the time, and for ERR, the hypothesis is false about 43% of the time.

The correlation of the index of smoothness with SEG is positive but not significant. The hypothesis that the index of smoothness increases as the number of segments in the flowgraph increases could be true but would be false about 25% of the time.

4.4.2. Analysis by Programming Languages.

64 of the selected algorithms are written in ALGOL. The remaining 14 are written in FORTRAN.

Assuming that the ALGOL flowgraphs constitute a parent population, the following hypothesis, Ho, can be tested on the basis of the index of smoothness of the flowgraphs:

Ho: The FORTRAN flowgraphs are a random sample group from the ALGOL
flowgraphs and there is no significant difference between the two groups.

A statistic, $Z$, is computed for the two groups as follows (14):

$$Z = \left( \frac{\sqrt{n} \left| M - m \right|}{S} \right)$$

where

- $n$ = numerical size of the sample group
- $m$ = mean of the index of smoothness of the sample group
- $M$ = mean of the index of smoothness of the parent population
- $S$ = standard deviation of the index of smoothness of the parent population.

Using the values shown in Table 9:

$$Z = \left( \frac{\sqrt{14} \left| 0.909 - 0.824 \right|}{0.150} \right)$$

$$= 2.12$$

This value of $Z$ indicates that the hypothesis, $H_0$, would be found to be true in less than 5% of the times it is tested. Therefore, there is a significant difference in the average index of smoothness of the FORTRAN group of flowgraphs and the average index of smoothness of the ALGOL flowgraphs.

This conclusion is supported by the correlation coefficients shown in Table 11 and Table 12. For the ALGOL flowgraphs, there is a negative, significant (at 0.1%) correlation between the index of smoothness and each of the following variables: ETC, LS, DETC, BSEG, NEST, DIFF and ALS. For the FORTRAN flowgraphs, the correlation between the index of smoothness and each of NEST and ALS, though negative as in ALGOL, is not significant at the 5% level. Also, for
the FORTRAN flowgraphs, the correlation between the index of smoothness and each of ETC, LS, DETC, BSEG and DIFF is negative and significant but at levels higher than 0.1%.

4.4.3. Analysis by "Age" of Algorithms

The selected algorithms were published over a period of thirteen years, from 1960 to 1972. Table 13 shows the mean index of smoothness for the selected algorithms published in those years. There is no discernible relationship between the mean index of smoothness and the length of time from the base year, i.e., it can not be said that the index of smoothness has increased or decreased with the passage of time.

4.4.4. Analysis by Regions.

The selected algorithms were grouped according to their geographical regions of origin. The ACM regions are listed in Table 14. Regions 1 - 11 are in continental America. Region 12 is Europe.

Table 15 shows the mean index of smoothness of the selected flowgraphs from each region. These indices are generally high. This is not surprising, since the algorithms were written by professionals.
<table>
<thead>
<tr>
<th>Serial # of Algorithms</th>
<th>001</th>
<th>002</th>
<th>015</th>
<th>017</th>
<th>018</th>
<th>019</th>
<th>020</th>
<th>023</th>
<th>025</th>
<th>028</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.000</td>
<td>1.000</td>
<td>0.727</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.546</td>
<td>0.976</td>
</tr>
<tr>
<td>ETC</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18</td>
<td>4</td>
</tr>
<tr>
<td>CR</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>ERR</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>LANG</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TIME</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>REG</td>
<td>04</td>
<td>07</td>
<td>07</td>
<td>11</td>
<td>07</td>
<td>06</td>
<td>04</td>
<td>07</td>
<td>02</td>
<td>04</td>
</tr>
<tr>
<td>LS</td>
<td>19</td>
<td>10</td>
<td>20</td>
<td>10</td>
<td>29</td>
<td>11</td>
<td>07</td>
<td>11</td>
<td>62</td>
<td>35</td>
</tr>
<tr>
<td>DETC</td>
<td>0.00</td>
<td>1.00</td>
<td>1.50</td>
<td>0.00</td>
<td>0.50</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>18.00</td>
<td>0.57</td>
</tr>
<tr>
<td>SEG</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>BSEG</td>
<td>09</td>
<td>07</td>
<td>17</td>
<td>03</td>
<td>15</td>
<td>04</td>
<td>04</td>
<td>03</td>
<td>60</td>
<td>22</td>
</tr>
<tr>
<td>NEST</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>DIFF</td>
<td>18</td>
<td>07</td>
<td>68</td>
<td>00</td>
<td>30</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>420</td>
<td>66</td>
</tr>
<tr>
<td>ALS</td>
<td>3.80</td>
<td>5.00</td>
<td>10.00</td>
<td>2.50</td>
<td>7.25</td>
<td>2.75</td>
<td>3.50</td>
<td>3.67</td>
<td>62.00</td>
<td>5.00</td>
</tr>
<tr>
<td>CERT</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
TABLE 2

Variables of Algorithms

<table>
<thead>
<tr>
<th></th>
<th>29</th>
<th>35</th>
<th>36</th>
<th>38</th>
<th>42</th>
<th>51</th>
<th>52</th>
<th>57</th>
<th>60</th>
<th>66</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETC</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CR</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>ERR</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LANG</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TIME</td>
<td>60</td>
<td>61</td>
<td>61</td>
<td>61</td>
<td>61</td>
<td>61</td>
<td>61</td>
<td>61</td>
<td>61</td>
<td>61</td>
</tr>
<tr>
<td>REG</td>
<td>04</td>
<td>03</td>
<td>03</td>
<td>03</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>LS</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>19</td>
<td>43</td>
<td>08</td>
<td>09</td>
<td>13</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>DETC</td>
<td>0.00</td>
<td>2.50</td>
<td>0.00</td>
<td>2.00</td>
<td>0.33</td>
<td>0.00</td>
<td>0.00</td>
<td>0.25</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SEG</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>BSEG</td>
<td>06</td>
<td>09</td>
<td>05</td>
<td>17</td>
<td>11</td>
<td>05</td>
<td>06</td>
<td>05</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>NEST</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DIFF</td>
<td>06</td>
<td>18</td>
<td>05</td>
<td>34</td>
<td>33</td>
<td>05</td>
<td>06</td>
<td>05</td>
<td>11</td>
<td>26</td>
</tr>
<tr>
<td>ALS</td>
<td>4.00</td>
<td>6.00</td>
<td>5.00</td>
<td>19.00</td>
<td>7.16</td>
<td>4.00</td>
<td>4.50</td>
<td>3.25</td>
<td>5.00</td>
<td>10.00</td>
</tr>
<tr>
<td>CERT</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>67</td>
<td>69</td>
<td>84</td>
<td>90</td>
<td>91</td>
<td>92</td>
<td>96</td>
<td>99</td>
<td>101</td>
<td>104</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>ETC</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.00</td>
<td>10.00</td>
<td>7.00</td>
<td>0.00</td>
<td>5.00</td>
<td>0.00</td>
<td>2.00</td>
</tr>
<tr>
<td>CR</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>4.00</td>
<td>11.00</td>
<td>0.00</td>
<td>0.00</td>
<td>3.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ERR</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
<td>4.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>LANG</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
<td>4.00</td>
<td>0.00</td>
<td>0.00</td>
<td>3.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>TIME</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>REG</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>LS</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DETC</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>SEG</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>BSEG</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>NEST</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DIFF</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ALS</td>
<td>7.00</td>
<td>7.00</td>
<td>2.33</td>
<td>2.33</td>
<td>5.00</td>
<td>8.00</td>
<td>3.00</td>
<td>3.60</td>
<td>9.00</td>
<td>0.00</td>
</tr>
<tr>
<td>CERT</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

TABLE 3
Variables of Algorithms
<table>
<thead>
<tr>
<th></th>
<th>105</th>
<th>109</th>
<th>115</th>
<th>143</th>
<th>148</th>
<th>151</th>
<th>153</th>
<th>155</th>
<th>158</th>
<th>160</th>
</tr>
</thead>
<tbody>
<tr>
<td>V a r i a b l e s o f A l g o r i t h m s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ETC</td>
<td>1.000</td>
<td>1.000</td>
<td>0.641</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.650</td>
<td>0.507</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>CR</td>
<td>1.000</td>
<td>1.000</td>
<td>0.641</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.650</td>
<td>0.507</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>ERR</td>
<td>1.000</td>
<td>1.000</td>
<td>0.641</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.650</td>
<td>0.507</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>LANG</td>
<td>1.000</td>
<td>1.000</td>
<td>0.641</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.650</td>
<td>0.507</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>TIME</td>
<td>1.000</td>
<td>1.000</td>
<td>0.641</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.650</td>
<td>0.507</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>REG</td>
<td>1.000</td>
<td>1.000</td>
<td>0.641</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.650</td>
<td>0.507</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>LS</td>
<td>1.000</td>
<td>1.000</td>
<td>0.641</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.650</td>
<td>0.507</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>DETC</td>
<td>1.000</td>
<td>1.000</td>
<td>0.641</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.650</td>
<td>0.507</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>SEG</td>
<td>1.000</td>
<td>1.000</td>
<td>0.641</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.650</td>
<td>0.507</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>BSEG</td>
<td>1.000</td>
<td>1.000</td>
<td>0.641</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.650</td>
<td>0.507</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>NEST</td>
<td>1.000</td>
<td>1.000</td>
<td>0.641</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.650</td>
<td>0.507</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>DIFF</td>
<td>1.000</td>
<td>1.000</td>
<td>0.641</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.650</td>
<td>0.507</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>ALS</td>
<td>1.000</td>
<td>1.000</td>
<td>0.641</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.650</td>
<td>0.507</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>CERT</td>
<td>1.000</td>
<td>1.000</td>
<td>0.641</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.650</td>
<td>0.507</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Variables of Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>164</td>
<td>176</td>
<td>188</td>
<td>189</td>
<td>195</td>
<td>211</td>
<td>212</td>
<td>214</td>
<td>215</td>
<td>216</td>
<td></td>
</tr>
<tr>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.482</td>
<td>1.000</td>
<td>0.814</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>ETC</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CR</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ERR</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>LANG</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>REG</td>
<td>04</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>09</td>
<td>06</td>
<td>10</td>
<td>12</td>
<td>07</td>
<td></td>
</tr>
<tr>
<td>LS</td>
<td>170</td>
<td>22</td>
<td>13</td>
<td>13</td>
<td>48</td>
<td>12</td>
<td>18</td>
<td>18</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>DETC</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.50</td>
<td>2.00</td>
<td>0.50</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>SEG</td>
<td>33</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>BSEG</td>
<td>23</td>
<td>09</td>
<td>11</td>
<td>11</td>
<td>28</td>
<td>09</td>
<td>45</td>
<td>11</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>NEST</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>DIFF</td>
<td>69</td>
<td>18</td>
<td>11</td>
<td>11</td>
<td>56</td>
<td>09</td>
<td>45</td>
<td>11</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>ALS</td>
<td>5.15</td>
<td>4.40</td>
<td>13.00</td>
<td>13.00</td>
<td>9.60</td>
<td>6.00</td>
<td>9.00</td>
<td>4.50</td>
<td>10.50</td>
<td></td>
</tr>
<tr>
<td>CERT</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>221</td>
<td>227</td>
<td>233</td>
<td>241</td>
<td>244</td>
<td>246</td>
<td>261</td>
<td>266</td>
<td>303</td>
<td>305</td>
</tr>
<tr>
<td>--------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>ETC</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>CR</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ERR</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LANG</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TIME</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>65</td>
<td>65</td>
<td>67</td>
<td>67</td>
</tr>
<tr>
<td>REG</td>
<td>06</td>
<td>12</td>
<td>06</td>
<td>12</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>05</td>
<td>12</td>
</tr>
<tr>
<td>LS</td>
<td>21</td>
<td>26</td>
<td>26</td>
<td>33</td>
<td>33</td>
<td>15</td>
<td>26</td>
<td>03</td>
<td>06</td>
<td>38</td>
</tr>
<tr>
<td>DETC</td>
<td>0.00</td>
<td>0.00</td>
<td>0.75</td>
<td>2.00</td>
<td>9.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.67</td>
</tr>
<tr>
<td>SEG</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>BSEG</td>
<td>07</td>
<td>17</td>
<td>19</td>
<td>30</td>
<td>31</td>
<td>07</td>
<td>24</td>
<td>01</td>
<td>04</td>
<td>34</td>
</tr>
<tr>
<td>NEST</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>DIFF</td>
<td>07</td>
<td>68</td>
<td>38</td>
<td>120</td>
<td>62</td>
<td>07</td>
<td>48</td>
<td>00</td>
<td>00</td>
<td>170</td>
</tr>
<tr>
<td>ALS</td>
<td>4.20</td>
<td>8.67</td>
<td>6.50</td>
<td>16.50</td>
<td>33.00</td>
<td>5.00</td>
<td>26.00</td>
<td>3.00</td>
<td>6.00</td>
<td>12.67</td>
</tr>
<tr>
<td>CERT</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>306</td>
<td>310</td>
<td>318</td>
<td>337</td>
<td>351</td>
<td>353</td>
<td>359</td>
<td>386</td>
<td>404</td>
<td>422</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>ETC</td>
<td>17</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>9</td>
<td>24</td>
<td>0</td>
<td>13</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>CR</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ERR</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LANG</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>TIME</td>
<td>67</td>
<td>67</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>69</td>
<td>69</td>
<td>70</td>
<td>70</td>
<td>72</td>
</tr>
<tr>
<td>REG</td>
<td>12</td>
<td>04</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>07</td>
<td>05</td>
<td>01</td>
<td>04</td>
<td>06</td>
</tr>
<tr>
<td>LS</td>
<td>51</td>
<td>18</td>
<td>74</td>
<td>22</td>
<td>37</td>
<td>49</td>
<td>21</td>
<td>45</td>
<td>39</td>
<td>21</td>
</tr>
<tr>
<td>DETC</td>
<td>17.00</td>
<td>0.33</td>
<td>1.00</td>
<td>0.00</td>
<td>2.25</td>
<td>4.80</td>
<td>0.00</td>
<td>6.50</td>
<td>7.50</td>
<td>1.00</td>
</tr>
<tr>
<td>SEG</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>BSEG</td>
<td>49</td>
<td>13</td>
<td>68</td>
<td>10</td>
<td>29</td>
<td>37</td>
<td>16</td>
<td>37</td>
<td>36</td>
<td>15</td>
</tr>
<tr>
<td>NEST</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>DIFF</td>
<td>196</td>
<td>26</td>
<td>198</td>
<td>10</td>
<td>87</td>
<td>111</td>
<td>48</td>
<td>111</td>
<td>108</td>
<td>30</td>
</tr>
<tr>
<td>ALS</td>
<td>51.00</td>
<td>6.00</td>
<td>24.67</td>
<td>5.50</td>
<td>9.25</td>
<td>9.80</td>
<td>10.50</td>
<td>22.50</td>
<td>19.50</td>
<td>7.00</td>
</tr>
<tr>
<td>CERT</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Variables of Algorithms</td>
<td>423</td>
<td>425</td>
<td>427</td>
<td>429</td>
<td>434</td>
<td>436</td>
<td>437</td>
<td>439</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ETC</td>
<td>3</td>
<td>9</td>
<td>23</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CR</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERR</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LANG</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>72</td>
<td>72</td>
<td>72</td>
<td>72</td>
<td>72</td>
<td>72</td>
<td>72</td>
<td>72</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REG</td>
<td>06</td>
<td>09</td>
<td>11</td>
<td>06</td>
<td>04</td>
<td>05</td>
<td>05</td>
<td>05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LS</td>
<td>34</td>
<td>34</td>
<td>32</td>
<td>29</td>
<td>57</td>
<td>14</td>
<td>17</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DETC</td>
<td>1.50</td>
<td>9.00</td>
<td>7.67</td>
<td>5.00</td>
<td>0.62</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEG</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSEG</td>
<td>31</td>
<td>32</td>
<td>25</td>
<td>27</td>
<td>36</td>
<td>10</td>
<td>13</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEST</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIFF</td>
<td>165</td>
<td>128</td>
<td>751</td>
<td>54</td>
<td>72</td>
<td>20</td>
<td>26</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALS</td>
<td>17.00</td>
<td>34.00</td>
<td>10.67</td>
<td>29.00</td>
<td>7.12</td>
<td>4.67</td>
<td>5.67</td>
<td>6.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CERT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>Sample Size</td>
<td>Non-Smooth</td>
<td>% Non-Smooth</td>
<td>Mean Index of Smoothness</td>
<td>Std. Deviation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>------------</td>
<td>--------------</td>
<td>--------------------------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALGOL</td>
<td>64</td>
<td>22</td>
<td>34.4</td>
<td>0.909</td>
<td>0.150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FORTRAN</td>
<td>14</td>
<td>7</td>
<td>50.0</td>
<td>0.824</td>
<td>0.223</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>78</td>
<td>29</td>
<td>37.2</td>
<td>0.893</td>
<td>0.167</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**TABLE 10**

Correlation of Index of Smoothness with Other Variables (All Selected Flowgraphs)

<table>
<thead>
<tr>
<th>Variable</th>
<th>ETC</th>
<th>CR</th>
<th>ERR</th>
<th>LS</th>
<th>DETC</th>
<th>SEG</th>
<th>BSEG</th>
<th>NEST</th>
<th>DIFF</th>
<th>ALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Size</td>
<td>78</td>
<td>40</td>
<td>40</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>78</td>
</tr>
<tr>
<td>Spearman Corr. Coeff.</td>
<td>-0.7524</td>
<td>-0.2365</td>
<td>-0.1288</td>
<td>-0.4888</td>
<td>-0.7294</td>
<td>0.1338</td>
<td>-0.5406</td>
<td>-0.4711</td>
<td>-0.5391</td>
<td>-0.4849</td>
</tr>
<tr>
<td>Signif.</td>
<td>0.001</td>
<td>0.142</td>
<td>0.428</td>
<td>0.001</td>
<td>0.001</td>
<td>0.243</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
</tr>
</tbody>
</table>
TABLE 11

Correlation of Index of Smoothness
with Other Variables
(ALGOL Flowgraphs)

<table>
<thead>
<tr>
<th>Variable</th>
<th>ETC</th>
<th>CR</th>
<th>ERR</th>
<th>LS</th>
<th>DETC</th>
<th>SEG</th>
<th>BSEG</th>
<th>NEST</th>
<th>DIFF</th>
<th>ALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Size</td>
<td>64</td>
<td>40</td>
<td>40</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Spearman Corr. Coeff.</td>
<td>-0.7495</td>
<td>-0.2365</td>
<td>-0.1288</td>
<td>-0.4225</td>
<td>-0.7297</td>
<td>0.1935</td>
<td>-0.5000</td>
<td>-0.4791</td>
<td>-0.5038</td>
<td>-0.4726</td>
</tr>
<tr>
<td>Significance</td>
<td>0.001</td>
<td>0.142</td>
<td>0.428</td>
<td>0.001</td>
<td>0.001</td>
<td>0.126</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
</tr>
</tbody>
</table>
### TABLE 12

Correlation of Index of Smoothness with Other Variables

*(FORTRAN Flowgraphs)*

<table>
<thead>
<tr>
<th>Variable</th>
<th>ETC</th>
<th>LS</th>
<th>DETC</th>
<th>SEG</th>
<th>BSEG</th>
<th>NEST</th>
<th>DIFF</th>
<th>ALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Size</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Spearman Corr. Coeff</td>
<td>-0.7574</td>
<td>-0.6798</td>
<td>-0.7618</td>
<td>-0.0632</td>
<td>-0.6445</td>
<td>-0.4286</td>
<td>-0.5545</td>
<td>-0.4670</td>
</tr>
<tr>
<td>Significance</td>
<td>0.002</td>
<td>0.007</td>
<td>0.002</td>
<td>0.830</td>
<td>0.013</td>
<td>0.126</td>
<td>0.040</td>
<td>0.092</td>
</tr>
<tr>
<td>Year</td>
<td>Sample Size</td>
<td>Non-Smooth</td>
<td>% Non-Smooth</td>
<td>Mean Index of Smoothness</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>------------</td>
<td>--------------</td>
<td>-------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1960</td>
<td>11</td>
<td>3</td>
<td>27.3</td>
<td>0.932</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>11</td>
<td>4</td>
<td>36.4</td>
<td>0.898</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>13</td>
<td>5</td>
<td>38.5</td>
<td>0.927</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>15</td>
<td>4</td>
<td>26.7</td>
<td>0.897</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>6</td>
<td>2</td>
<td>33.3</td>
<td>0.929</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>2</td>
<td>0</td>
<td>0.0</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>5</td>
<td>4</td>
<td>80.0</td>
<td>0.789</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>1</td>
<td>0</td>
<td>0.0</td>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>3</td>
<td>2</td>
<td>66.7</td>
<td>0.906</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>2</td>
<td>2</td>
<td>100.0</td>
<td>0.540</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>9</td>
<td>3</td>
<td>33.3</td>
<td>0.859</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>78</td>
<td>29</td>
<td>37.2</td>
<td>0.893</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Northeast</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Greater New York</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Allegheny</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Capital</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Southeast</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>East Central</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>North Central</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>South Central</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Mountain</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Northwest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Southern California</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Europe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TABLE 15
Analysis by Regions of Origin of Algorithms

<table>
<thead>
<tr>
<th>ACM Region Code</th>
<th>Sample Size</th>
<th>Non-Smooth</th>
<th>% Non-Smooth</th>
<th>Mean Index of Smoothness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>100.0</td>
<td>0.769</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>100.0</td>
<td>0.546</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>4</td>
<td>40.0</td>
<td>0.880</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>6</td>
<td>54.5</td>
<td>0.816</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>0</td>
<td>0.0</td>
<td>1.000</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>1</td>
<td>12.5</td>
<td>0.935</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>2</td>
<td>28.5</td>
<td>0.937</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>100.0</td>
<td>0.886</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>1</td>
<td>50.0</td>
<td>0.833</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0</td>
<td>0.0</td>
<td>1.000</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>3</td>
<td>30.0</td>
<td>0.888</td>
</tr>
<tr>
<td>12</td>
<td>18</td>
<td>8</td>
<td>44.4</td>
<td>0.908</td>
</tr>
<tr>
<td>TOTAL</td>
<td>78</td>
<td>29</td>
<td>37.2</td>
<td>0.893</td>
</tr>
<tr>
<td>1-11</td>
<td>60</td>
<td>21</td>
<td>35.0</td>
<td>0.889</td>
</tr>
</tbody>
</table>
CHAPTER 5

CONCLUSION

In this research, an index of smoothness has been defined for computer program flowgraphs. The summary of the research, presented in this chapter, deals with a recapitulation of the central concepts, comments on the results obtained and assessments of possible applications, limitations and extensions.


The definition of the index of smoothness of a given program flowgraph is based on the graph grammar, SG, defined in Chapter 2. If, by applying the productions of SG, the given flowgraph can be parsed successfully, then, the flowgraph is smooth; otherwise, it is non-smooth. A smooth flowgraph is assigned an index of smoothness of 1; a non-smooth flowgraph is assigned an index of smoothness which is less than 1.

A modified form of Cocke's "interval," the "full interval" is used in the parsing process.

Non-smooth sub-flowgraphs of a given non-smooth flowgraph are identified and transformations are performed on these non-smooth sub-flowgraphs to obtain a smooth flowgraph which is "equivalent" to the given non-smooth flowgraph.

It is shown that the transformations on non-smooth flowgraphs have the effect of increasing the cardinality of the arcs of the flowgraph. The index of smoothness is obtained by comparing the
cardinalities of the arcs of the original flowgraph and the final flowgraph.

5.2 Results.

The index of smoothness has been derived for each of seventy-eight algorithms randomly selected from CACM. Other variables, described in Chapter 4, were also measured for each algorithm. A statistical analysis of the relationship between the index of smoothness and these variables give the following results:

There is a significant, negative correlation between the index of smoothness and each of the following variables:

- the number of explicit transfers of control
- the average number of explicit transfers of control in a segment
- the level of nesting of the innermost loop or alternation
- the number of nodes in the minimal flowgraph
- the number of nodes in the biggest segment
- the average number of nodes in a segment

If the index of smoothness is regarded as a figure of merit, then, the negative correlation between the index of smoothness and the average number of explicit transfers of control in a segment is a confirmation of Dijkstra's suspicion (2) that "the quality of programmers is a decreasing function of the density of goto statements in the programs they produce."

The negative correlation between the index of smoothness and the level of nesting of the innermost loop or alternation implies that, for the selected algorithms, the deeper the level of nesting of loops
and alternations, the smaller the index of smoothness tends to be. This can be explained by noting that in many cases, "escape" exits are required from inner to outer levels and these escapes are usually effected by use of explicit transfers of control and in a manner which is ungrammatical in the grammar SG.

The negative correlations between the index of smoothness and the average number of nodes in a segment and the negative correlation between the index of smoothness and the number of nodes in the biggest segment suggest that the index of smoothness tends to be higher for flowgraphs with many segments and fewer nodes per segment. If the logic of the flowgraph is concentrated in a few segments, then, the level of nesting in one or more of those few segments tends to be deep. As explained above, deep levels of nesting tend to result in a lower index of smoothness.

For the certified algorithms, there is no significant correlation between the index of smoothness and the number of critical remarks of the certifiers; neither is there a significant correlation between the index of smoothness and the number of errors detected in the algorithms. This implies that in many cases, the critical remarks and errors are not closely related to the structure of the flowgraph but may be related to some other variables.

There is a significant difference between the index of smoothness of the average ALGOL flowgraph and the index of smoothness of the average FORTRAN flowgraph. The average FORTRAN flowgraph has a lower index of smoothness.
If the index of smoothness is regarded as a figure of merit, then, there is no convincing evidence that the programming style of the professional computing community has improved or deteriorated over the thirteen-year period from 1960 - 1972

5.3. Applications

The index of smoothness and the program, SMOOTH, used to derive it can be useful in an educational environment. Given a number of programs computing the same function, the index of smoothness of each program can be derived. A subset of the programs scoring the highest indices can be chosen as the best. The index of smoothness can also be used to rank such programs.

The monitoring facilities provided in program SMOOTH can be used to identify those sections of a student's program flowgraph which are non-smooth. Such non-smooth sections of his flowgraph can reveal basic misunderstandings or inadequacies on the part of the student. More explanations and instructions can then be supplied to the student to enable him to correct these deficiencies and produce better structured programs.

In addition to calculating the index of smoothness, the independent variables which tend to decrease the index of smoothness, such as the number of explicit transfers of control, deepest level of nesting of loops and alternations, etc., can be measured for students' programs. An analysis of these data can reveal defects in the students' understanding of the most suitable language constructs to use in certain circumstances; for example, if a student's program scores an index of smoothness of 1 and this program contains a substantial number of
explicit transfers of control, then, these explicit transfers of control could be avoided by the use of if-then-else and/or loop-control constructs. It may be that the student does not fully understand the appropriate uses of these control constructs.

A smooth program is easy to read and understand. Control always flows forward, from one segment to another, from program beginning to the end. In partitioning a big program into sections for overlay in a real-storage environment, or into pages in a virtual-storage environment, one of the main problems is determining the boundaries between one section/page and another. This problem is less acute for smooth programs. The segments of a smooth program are ideally suitable units. Consecutive small segments can be packed into a single page or a large segment can be subdivided into a family of pages such that each page has a single entry and a single exit. Automatic and efficient pagination for smooth programs is thus feasible if the object programs and the corresponding flowgraph are available to the paging system. If a given program is non-smooth, the program SMOOTH can be used to indicate the transformations necessary to convert it to a smooth program.

Compilers of higher-level languages usually analyze the control flow of programs to be compiled. For purposes of optimization, a determination of the local and global attributes of variables is necessary. Automatic control flow analysis and determination of local and global attributes of variables are greatly simplified if a program's flowgraph is smooth. By ensuring that a program is smooth before it is presented for compilation, the benefits of fast compilation and efficient object code are obtained.
The term "structured programs" has, up until now been in use within the computing community without a precise definition of the term. Now, a precise definition of "smooth programs" has been made in this research and the term "smooth programs" can be used to convey a precise meaning.

5.4 Limitations and extensions

It is necessary to construct the flowgraph of a program whose index of smoothness is required. The flowgraph is then coded into a multi-linked list as explained in Section 4.2.2. The input to program SMOOTH is the coded form of the flowgraph. At present, the construction of the flowgraph and its coding into input form are performed manually. It would be an improvement to develop a pre-processing program which would accept a source-language program and automatically construct its flowgraph and generate the equivalent list of nodes which would then form the input to program SMOOTH.

Program SMOOTH displays the input flowgraph, and if it is non-smooth, also displays the transformed smooth equivalent flowgraph. These displays are currently in the form of a multi-linked list of the nodes of the flowgraphs. For rapid visual inspection and comparison, actual flowgraphs are preferable. A sub-program to display actual flowgraphs can be added to program SMOOTH.

Dr. L. J. White of the Department of Computer and Information Science, Ohio State University has suggested that there might be a connection between the index of smoothness of a flowgraph and the coplanarity of the flowgraph. This suggestion is worthy of further investigation.
If Larry Weissman (12) succeeds in defining an index of psychological complexity of computer programs, then, it would be possible to correlate the index of smoothness and the index of psychological complexity. Such correlation might reveal the properties that a program should have to make it easily comprehensible to a human as well as to a computer.
APPENDIX

SUMMARY OF PROGRAM SMOOTH

The flowgraph of the program is shown in Chart XXV. The instructions for operating the program are given in Section 4.2.

The program is written in PL/I and consists of a main procedure and 25 embedded procedures.

SMOOTH: Procedure Options (Main);
ISONLIST: Procedure (Lstart, Lident, Adrsee, Boov, Tag);
/* Lstart, Adrsee and Tag are pointers, Boov is a boolean variable and Lident is an integer code representing the structure of each element of the list */
Boov = false and Tag = NULL:
Search through the list for Adrsee, beginning at Lstart;
If found
then do; Boov = true; Tag = address of the location;
  end;
return;
end ISONLIST:

REMOVE: Procedure (Tag);
/* remove from DBTLST, a doubly-linked list, the element whose address is given in TAG */
if the element is the leading element
then DBTLST = address of next element in the list;
else do; if the element is last in the list
107
then terminate list at second to last;
else adjust pointers in previous and
   succeeding elements of Tag;
end;
end REMOVE:
INTAVALS: Procedure (Flograf);
/* the address of the entry-node of a flowgraph is given in
   Flograf. Partition the flowgraph into intervals. A node
   belongs to an interval if all of its immediate predecessors
   are in the interval */ starting at HDRLST, construct a list
   of HEADERS; indicate in each header-structure the number of
   nodes in the interval, the number of exits, address of the
   exit-node (if single), a pointer to the list of nodes in the
   interval and indicate whether interval is cyclic;
   if any of trace options 1 and 2 is exercised
   then output required information;
end INTAVALS:
CREATE: Procedure (Pred, Succ):
   /* create a new node between Pred and Succ */ assign the next
      higher identification number to the node;
      if either Pred or Succ is non-null
         then adjust pointers to neighbour nodes as necessary;
end CREATE;
CHKCYCLC: Procedure (Ntrynode, Exitnode, Ntvlst, Itrtnode, Modify);
   /* check if a given cyclic subgraph has a single valid exit.
      Ntvlst heads a list of nodes in the subgraph. Itrtnode
is the address of the apparent iteration node */ Obtain the
loop-set; determine the exits from the loop and the
respective target nodes;
if there is a single exit from the loop
then do; if the apparent iteration node does not
post-dominate every other node in
loop-set
then Modify, Snglnvald = '1'B;
else Modify, Snglnvald = '0'B;
end;
else do; Modify = '1'B; Snglnvald= '0'B; end;
if Modify
then do; transform subgraph so as to have a single valid
exit; Match "old" exit to respective target nodes;
    Anytrans = '1'B;
end;
if trace option 3 is requested
then output the loop-set, the exits and target nodes;
end CHKCYCLC:

REDUCE: Procedure (Nytrynode, Exitnode, Cyclic);
    /* reduce the interval whose entry-and-exit-nodes are given */
    create a new node to replace the interval; the predecessors
    and successors of the new nodes are those of the interval
    it replaces; the new node points to Nytrynode and Exitnode;
    the pointer, in Nytrynode, to "external" successor are
    changed to point to the newly created node;
end REDUCE;
**ENUMRATE:** Procedure (Grafentr, Enumhd, Cardinal);

/* construct a linear list of all the nodes of a
    flowgraph whose entry-node is Grafentr; Enumhd points to
    the head of the list; Leave a count of all the arcs of the
    flowgraph in Cardinal */ Cardinal = 0; Put address of
    Grafentr in first element; put address of that element in
    Enumhd; sequence through the list, starting from Enumhd;
    do while not at end of list;
        obtain successors of current node; add the number of
        non-null successors to Cardinal; do for each non-null
        successor;
            if its address is not already on the list
                then add its address to end of the list;
            end;
        end;
        current element = next in sequence;
    end;
end ENUMRATE;

**DSPLGRAF:** Procedure (Entnode, Arcount);

/* Display the nodes of a flowgraph, whose entry-node address
    is at Entnode. Set Arcount equal to the cardinality of the
    arcs of the flowgraph */
print headings; Call Enumrate (Entnode, Nodlst, Arcount);
do for each element of Nodlst;
    print identification number, predecessors and successors
end;
free Nodlst;
end DSPLGRAF;

DSPLIST: Procedure (Lsthd, Lident);

/* Display elements of a list whose head-address is Lsthd and
   element structure code is Lident */ sequence through list,
   starting from Lsthd, obtaining identification number of
   node whose address is in the current list element; arrange
   these 10 to a line; print;

end DSPLIST;

RECOVER: Procedure (Flohdr);

/* Recover a flowgraph from its reduced state*/ call procedure
PRESUN to make copies of reduced intervals which are
referenced more than once; zigzag through the nodes of the
flowgraph and replace each reduced node by the interval it
represents;
if trace-option 6 is exercised
then display every node representing a reduced interval;
end RECOVER;

CONSECOL: Procedure (Consec, Confirm);

/* Confirm if two nodes whose addresses are given in Consec are
   members of a set of consecutive collector nodes */
if one of the pair is either an immediate predecessor or
immediate successor of the other
then confirm

end CONSECOL;
CHKALTRN: Procedure (Ntrynode, Exitnode, Itrtnode, Cyclic, Fulcyclic);

/* Addresses of entry-, exit- and iteration-nodes of a
subgraph are given in Ntrynode, Exitnode and Itrtnode
respectively. If Cyclic is true, then the subgraph is
cyclic. If Fulcyclic is true, then the cyclic subgraph has
a single exit */

make a list, Altlst, of all decision nodes, excluding any
iteration node, in the subgraph;
do for each member of Altlst;
  if it is the head of an elementary decision element
  then do; if the elementary decision element is an
  alternation
    then do; shrink the alternation to a single
    node; remove the decision node from
    membership of Altlst;
  end;
  else do; identify the valid collector node
          of the decision element;
          do for each of the invalid collector
          nodes in the decision element; copy
          the path from the invalid collector
          to the valid collector node;
  end;
end;

end;
end;

replace each shrunken node by the corresponding alternation;
end CHKALTRN;

CHKPATH: Procedure (Decsn, Colec, Nodadres, Prex, Pathed, Intvhd, Shrnkhd, Newhd);

/* Check the path from pathed until either a decision node or
collector node is found. Put address of found node in
Nodadres. Put address of its predecessor on the path in
Prex. If a decision node is found, set Decsn to true
else set Colec to true */
Current = Pathed; Inbound = '1'B;
Decsn, Colec = '0'B; Nodadres, Prex = Null;
do while (Inbound and 7 Decsn and 7 Colec);
if current is not in this interval
  then do; if current is not a shrunken node
    then do; if current is not a newly created node
      then Inbounds = '0'B;
        end;
    end;
  end;
else do; if both successors of current are non-null
  then do; Decsn = '1'B; Nodadres = Current; end;
else if both predecessors of current are non-null
  then do; Colec = '1'B; Nodadres = Current; end;
else Current = successor of current;
end;
end CHKPATH;
SHRINK: Procedure (Decision, Collector):

/* Replace an alternation with a single node */ Allocate a
new node; Adjust pointers such that the previous predecessor
and successor of the alternation are now the predecessor and
successor of the new node; Set pointers associated with the
new node to point to the alternation if replaced; Adjust
pointers in the decision and collector nodes of the
alternation to point to the new node which replaced the
alternation;

end SHRINK;

EXPAND: Procedure (Shrnknod);

/* Expand the node whose address is Shrnknod, into an
alternation*/
Access the node to obtain address of the alternation it replaced;
Connect predecessor and successor of the node to the entry and
exit, respectively, of the alternation; Free Shrnknod;

end EXPAND;

REARANGE: Procedure (Pnod, Adjcnt);

Relocate one of two collector nodes in a subgraph so that an
alternation with one entry and one exit can be recognized;

end REARANGE;

COPYPATH: Procedure (Pvec, Newhd, Prenum, Shrn, Lshrn);

/* Copy the path from an invalid collector node to the valid
collector node. Pvec is an 8-element vector of addresses
of the valid and invalid collector nodes and their
predecessors and successors */
Copy the nodes from the invalid collector node whose address is in Pvect (2) to the valid collector node addressed by Pvect (1); Effectively copy the arcs on the path by appropriate settings of the pointers of the new nodes; Move the invalid collector node to the end of the copied path;

end COPYPATH;

PRESCAN: Procedure (Flotop);

/* Scan through a flowgraph, whose entry is addressed by Flotop, for reduced intervals which are referred to at more than one point in the flowgraph. Make a copy of the interval for each reference other than the first */
do for every node in the flowgraph;
    if the current node is a reduced interval
ten then do; access the interval;
        if the interval header does not point to the reduced node
then call Minicopy;
end PRESCAN:

MINICOPY: Procedure (Xpnod, Pcode, Schst, Shlst);

/* Make a copy of a subgraph which has been reduced to a single node. Pcode = 1 if the subgraph is an alternation, else the subgraph is an interval */
Obtain from Xpnod the addresses of head and tail of the subgraph; Make a list of the nodes from head to tail; for each node on the list, allocate a new node; make a pair-list for addresses of old and new nodes; Use the pair-list to set
pointers to neighbor nodes;

If a node of the copied subgraph represents a reduced subgraph
then add address of the node to Xpandlst, a list of subgraphs
to be copied;

if the address of a copied node is on Shlst list
then add address of its copy to Shlst list;
end MINICOPY;

UNTANGLE: Procedure;

/* Identify a maximal, strongly connected region containing a loop
or loops with multiple entries. Select, from among a set of
multiple entries to a loop, a preferred entry. Remove one
of the other entries by path-splitting. The flowgraphs
intervals are already known. Addresses of the interval
headers are on HDRLST */
call HIAPRED (Hipreds);

identify an interval header which is also a collector node;

if the collector node is an entry to a loop
then the higher immediate predecessor is external to the loop and
the other predecessor is in the loop;
call ISALOOP to confirm or deny that the collector node is in a
loop; if confirmed
then do; the loop thus confirmed is a maximal strongly connected
region; identify within the region a loop with multiple entries;
call PREFENTR to identify the preferred entry;
call REMOVENT to remove one of the entries;
end;

end UNTANGLE;

HIAPRED: Procedure (Predlst);

/* Put in a list, addressed by Predlst, the address of the
higher of the two predecessors of each interval header
node which is also a collector node */

Scan through the nodes of the flowgraph in their order of
precedence, starting from flowgraph entry node;

if a successor, S, of the current node is a collector node
then do; if S is also on the header list

then the current node is the higher predecessor of S;

end;

if trace option 5 is exercised

then print each header-collector and its higher predecessor;

end HIAPRED;

ISALOOP: Procedure (Potentry, Potlatch, Outside, Looplst,

Hedacols, Confirm);

/* Confirm if there is a loop whose entry-node address is
Potentry and whose latching-node address is Potlatch. A
list of nodes which are known to be outside the loop is
addressed by Outside. If the loop is confirmed, the nodes
inside the loop are listed in Looplst. The collector nodes
in the loop which are also on the list of interval headers,
are listed in Hedacols */
Starting from Potlatch, scan the flowgraph backwards, i.e., from
a node to its predecessors; stop scanning on a path which leads
Outside; if any of the scanning paths should lead to Potentry
then a loop is confirmed; if the loop is confirmed, scan
forward from Potentry to Potlatch; the nodes which are not
outside and which are reached on both backward and forward
scans are added to Looplst;
if trace option 5 is exercised
then do; display Looplst; display Hedacols; end;
end ISALOOP;

PREFENTR: Procedure (Loophd, Ntrylst, Prentry);

/* Put in Prentry the address of the preferred entry to a loop.
The addresses of the nodes of the loop are in a list addressed
by Loophd. The addresses of the entry nodes of the loop are
in a list addressed by Ntrylst */

identify the exits of the loop;
if there is only one valid exit from the loop
then the entry which is the closest predecessor of the valid
exit is chosen as the preferred entry;
else do; the entry which, if chosen as the preferred one,
   requires the minimum path-copying from the other entries
is selected as the preferred entry; the loop is transformed
to have only one exit;
if trace option 5 is exercised
then display the preferred entry;
end PREFENTR;

SIMPATHS: Procedure (Porigin, Destin, Loophd, Plist, Pknt);
/* Construct a list, addressed by Plist, of the nodes
between Porigin and Destin but within the loop whose nodes
addresses are contained in a list addressed by Loophd.
Leave a count of the nodes in Plist in Pknt */
Starting from Destin, move backwards until Porigin is reached;
reject any node encountered which is not in the loop list;
if all the back paths terminate on Porigin
then Plist is complete;
else do; move forward from Porigin until Destin, rejecting
any node encountered which is not in the loop list; the
nodes required in Plist are those reached on both the
backward and forward moves;
end;
end SIMPATHS;

REMOVENT: Procedure (Loophd, Prentry, Others);
/* Remove one of the multiple entries to a given loop.
Address of the preferred entry to the loop is Prentry.
Addresses of other entries are given in a list addressed
by Others */
Count the number of nodes in the loop; this is the upper limit to
the number of nodes to be copied in the process of removing an
unwanted entry; Select one of the unwanted entries for removal;

If the entry node and the preferred entry node are among a set of consecutive collector nodes then the two entries are effectively identical;
else do; identify all the successor nodes from the unwanted entry to the preferred entry;
make a copy of each of those nodes; identify and connect the predecessors and successors of the new nodes;
if trace option 5 is exercised then display the unwanted entry and the nodes which are copied to effect its removal;
end REMOVENT:

(Main Procedure follows)
read number of runs and option-flag;
if option flag is non-zero then read global trace options for the runs;
iter: do for each run;
read number of nodes in flowgraph, option flag and name of flowgraph;
if option flag is non-zero the read local trace options to replace the global options;
read the ident. number, the predecessors and successors of each node of the flowgraph;
display the nodes specifications as read in;
check validity of the nodes as specified;
if any node is invalid
then output detailed error message; else
analy: do; determine and display the cardinality of the arcs of
the given flowgraph;
complete = '0'B; nthorder = 0;
korder: do while not complete;
partition flowgraph into intervals;
if there is only one interval
then complete = '1'B;
do for each full interval;
if the interval is cyclic
then call CHKCYCLC:
call CHKALTRN;
call REDUCE;
end;
if the nthorder flowgraph is not reducible
then call UNTANGLE;
nthorder = nthorder + 1;
end korder;
if there were no transformations to make lowgraph smooth
then the index of smoothness = 1;
else do; call RECOVER (Floptr);
call DSPLGRAF (Floptr, Cadnalaf);
calculate the index of smoothness
end;

display the index of smoothness;

end analy;

decrease the number of runs by 1;

end iter;

end SMOOTH;
BIBLIOGRAPHY


