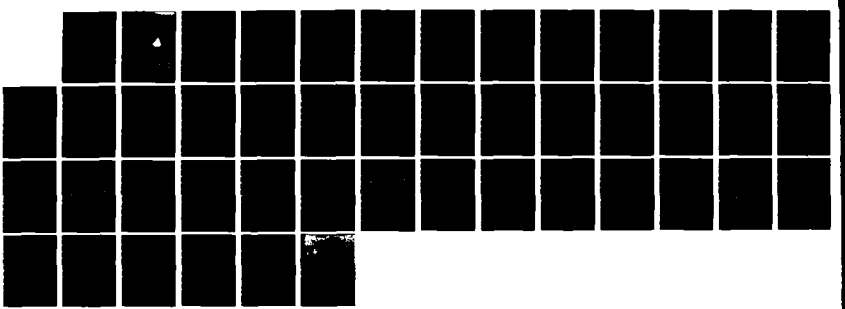
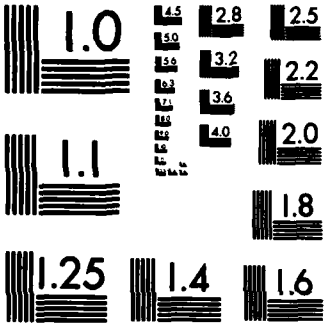
AD-A134 190 A COMPUTER PROGRAM FOR RELIABILITY EVALUATION OF LARGE-SCALE UNDIRECTED N..(U) CALIFORNIA UNIV BERKELEY OPERATIONS RESEARCH CENTER M G RESENDE OCT 83 1/1

UNCLASSIFIED ORC-83-10 DAAG29-81-K-0160 F/G 9/2 NL





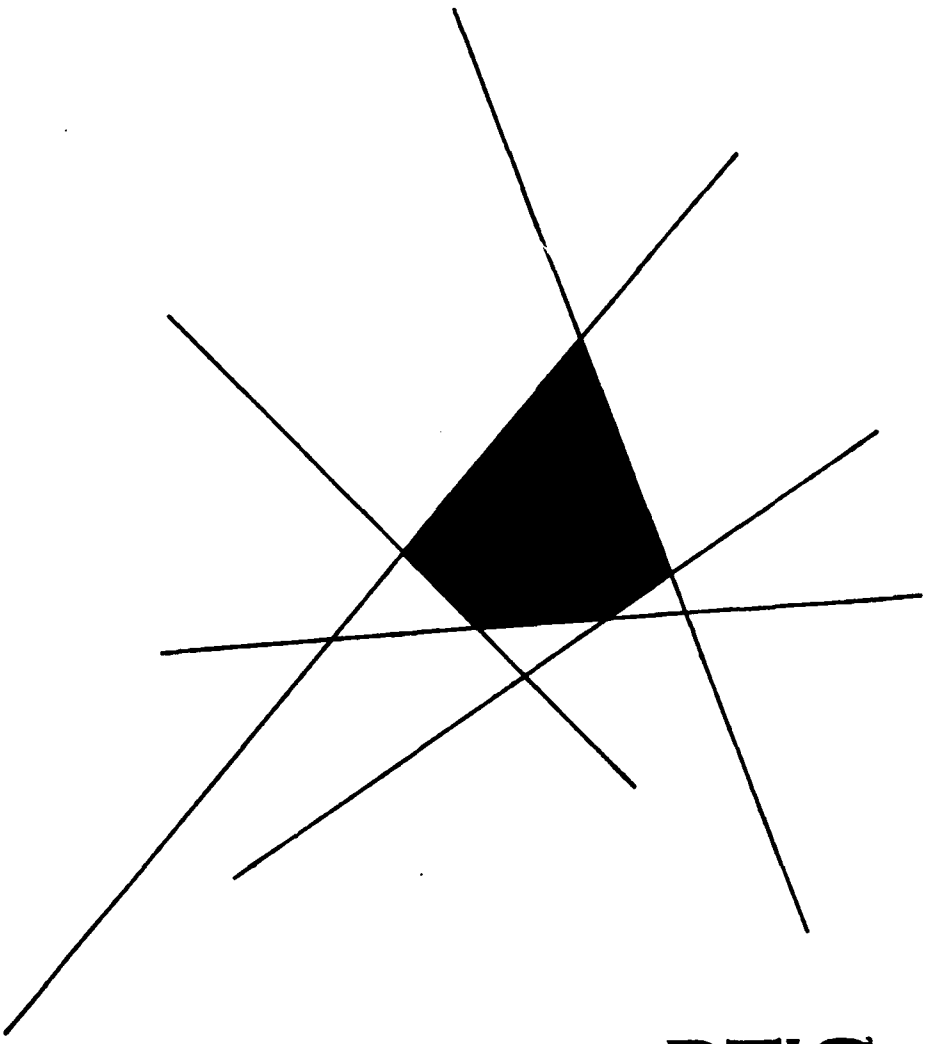
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

A COMPUTER PROGRAM FOR RELIABILITY EVALUATION OF LARGE-SCALE UNDIRECTED NETWORKS VIA POLYGON-TO-CHAIN REDUCTIONS

(12)

AD-A134 190

by
MAURICIO G. C. RESENDE



OPERATIONS RESEARCH CENTER

UNIVERSITY OF CALIFORNIA · BERKELEY

P o l y C h a i n

A COMPUTER PROGRAM FOR RELIABILITY EVALUATION OF LARGE-SCALE UNDIRECTED NETWORKS VIA POLYGON-TO-CHAIN REDUCTIONS[†]

Operations Research Center Research Report No. 83-10

Mauricio G. C. Resende

October 1983

U. S. Army Research Office - Research Triangle Park

DAAG29-81-K-0160

Operations Research Center
University of California, Berkeley

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED.

[†]Partially supported by Conselho Nacional de Desenvolvimento Cientifico e Tecnologico - CNPq, Brazil. Reproduction in whole or in part is permitted for any purpose of the United States Government.

THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF THE AUTHOR(S) AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.

GE	READ INSTRUCTIONS BEFORE COMPLETING FORM
ORT ACCESSION NO. A134190	3. RECIPIENT'S CATALOG NUMBER
VALUATION VIA POLYGON-	5. TYPE OF REPORT & PERIOD COVERED Research Report
	6. PERFORMING ORG. REPORT NUMBER
	8. CONTRACT OR GRANT NUMBER(s) DAAG29-81-K-0160
	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS P-18195-M
	12. REPORT DATE October 1983
na 27709	13. NUMBER OF PAGES 41
Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

ution unlimited.

k 20, if different from Report)

fy by block number)

Computer Program
Data Structures

by block number)

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT

This report discusses the design and implementation of PolyChain, a FORTRAN program for reliability evaluation of large-scale undirected networks via polygon-to-chain reductions.

First, theoretical results presented by Satyanarayana and Wood (1982) are briefly discussed. Then, the program's design and its implementation in FORTRAN are described in a system's manual. A user's guide contains instructions on problem solving and output interpretation. Finally, some large scale problems are tested to evaluate the code's performance and capabilities.

P o l y C h a i n

A Computer Program for Reliability Evaluation of Large-Scale Undirected Networks via Polygon-to-Chain Reductions

by

Mauricio G. C. Resende

Operations Research Center
University of California, Berkeley

1. Introduction

The evaluation of network reliability [A83], is of great importance in engineering. It is both an important design and operations parameter in such systems as communications, power, and computer, to name a few.

Satyanarayana and Wood [S82], [W82] introduced a linear time algorithm for computing the reliability of a network with an underlying series-parallel structure. This algorithm employs the new polygon-to-chain reductions. They also present an extension to the algorithm which enables the generation of a reduced network when the input network is not totally reducible.

This report discusses the design and implementation of PolyChain (Polygon-to-Chain), a portable FORTRAN program for K-terminal network reliability evaluation via polygon-to-chain reductions. Doubly linked multilists and linear stacks are used in this implementation of the algorithm in such a way that large-scale undirected networks are treated efficiently. PolyChain is modular, thoroughly commented, has input data consistency tests, and is structured whenever possible, to facilitate maintenance. There are four output options, which produce flexible, informative reports.

Section 2 briefly discusses some of the theoretical results of polygon-to-chain reductions. In section 3, a system's manual describes the algorithm's implementation in FORTRAN. Section 4 consists of a user's guide to PolyChain. In section 5, a number of large-scale problems problems are tested on PolyChain, illustrating the code's performance capabilities. Conclusions and recommendations are made in section

6. References are in section 7.

2. Polygon-to-Chain Reductions and Series-Parallel Graphs

For a complete discussion of polygon-to-chain reductions and series-parallel graphs, see [S82], [W82]. In this section, the K-terminal network reliability problem will be defined. Basic definitions will be made so that the algorithm of Satyanarayana and Wood can be presented.

2.1. K-Terminal Reliability

Let G=(V,E) be a graph, where V and E are respectively G's vertex set and edge set. We assume that vertices function perfectly and that edges have a probability of functioning, which may be less than 1. Edge e(i) has probability p(i) of functioning and q(i) = 1 - p(i) of not functioning. Associated with G is a subset of V denoted by K. These distinguished vertices are called the K-vertices of G. G(K) is graph G with K specified. The K-terminal reliability of G(K), R[G(K)], is the probability that, at a given time t, all K-vertices are connected by working edges.

2.2. Simple Reductions

To compute the K-terminal reliability of G, it is desirable to reduce the size of G. There are three well known simple reductions: parallel reduction, series reduction and degree-2 reduction. In parallel reduction, parallel edges e(a)=(u,v) and e(b)=(u,v) are replaced by edge e(c)=(u,v) with edge probability p(c) = 1 - q(a)q(b). In series reduction , edges e(a)=(u,v) and e(b)=(v,w) are replaced by edge e(c)=(u,w) with probability p(c) = p(a)p(b). Let u,v, and w be edges in set K. Let deg(v) = 2 and consider two edges e(a)=(u,v) and e(b)=(v,w), such that u is not equal to w. A degree-2 reduction substitutes edges e(a) and e(b) by edge e(c)=(u,w) with p(c)=p(a)p(b)/[1-q(a)q(b)] and R[G(K)]=[1-q(a)q(b)]R[G(K-v)], where G(K-v) is the new reduced graph. These simple reductions are examples of reliability-preserving reductions , where a graph G is replaced by a reduced graph G', and R(G)=OMEGA*R(G'), where OMEGA depends on the reduction.

2.3. Series-Parallel Reducible and Complex Graphs

A series-parallel graph is a graph that can be reduced to a tree after successive series and parallel reductions. For a given series-parallel graph G, G(K) may or may not be reduced to a single edge by successive simple reductions. This will depend on the elements of set K. G(K) is series-parallel reducible if it can be reduced to a single edge by successive simple reductions. It is series-parallel complex if it is not series-parallel reducible.

2.4. Chains and Polygons

A chain X is an alternating sequence of distinct vertices and edges, such that all vertices, except for the two endpoints, have degree 2. Let X1 and X2 be two distinct chains with common endpoints. The union of X1 and X2 constitutes a polygon.

2.5. Polygon-to-Chain Reductions

The main result in [S82], [W82] follows. Let G(K) be a graph that admits no simple reductions. If G(K) contains a polygon, it is one of the eight types. A reliability preserving reduction permits the replacement of the polygons by chains with the new edge reliability computed according to given formulas.

2.6. An O(|E|) Algorithm for Series-Parallel Complex Graphs

An O(|E|) algorithm for a nonseparable series-parallel complex graph is presented in [S82], [W82]. PolyChain is a direct implementation of that algorithm utilizing extension 4' in place of step 4.

3. System's Manual

In this section, the code is briefly described. First, programming is commented and data structures discussed. Next, data consistency tests are considered. A data dictionary is presented. The program's COMMON blocks and all procedures are described, along with their corresponding inputs and outputs. Finally, we comment on the code's debugging feature.

3.1. Program Design and Implementation

3.1.1. Programming

PolyChain is designed to be portable. All algorithmic routines are written in standard FORTRAN IV. Only output related code is system dependent. The code is intended to be friendly to both the person using it in an application and the one extending or maintaining it. The program is modular, has format free input and informative outputs.

3.1.2. Data Structures

The algorithm of Satyanarayana and Wood manipulates undirected networks. These networks can be represented in many ways in a computer code. Matrix representation has many drawbacks. These matrices are, in practice, extremely sparse. Densities of less than one percent are not uncommon. Matrix representation is inefficient both with respect to core usage and execution times. For a static network, an efficient representation is a packed matrix. In the algorithm considered here, the network is dynamic. Helgason and Kennington [H80], Thesen [T78], Berztiss [B75], among many, discuss efficient network representations using linked list data structures. Thesen [T78] and Berztiss [B75] discuss the implementation of linked lists in FORTRAN.

During the algorithm's reduction process, one or more edges or vertices are removed from the network. In the data structure, this corresponds to removing elements from the lists. This process is repeated frequently in the algorithm. Doubly linked lists [K73],[B75], require more core, but are more efficient when many element deletions are required. They are implemented in PolyChain. Each vertex has a list of adjacent vertices, which besides indicating which vertices are adjacent to it, also provides information determining whether the vertex and its adjacent vertices belong to set K. For every element of the list, there is a pointer indicating the address where information about the edge, corresponding to these two vertices, is kept. Figure I illustrates this multilist data structure for a small

network.

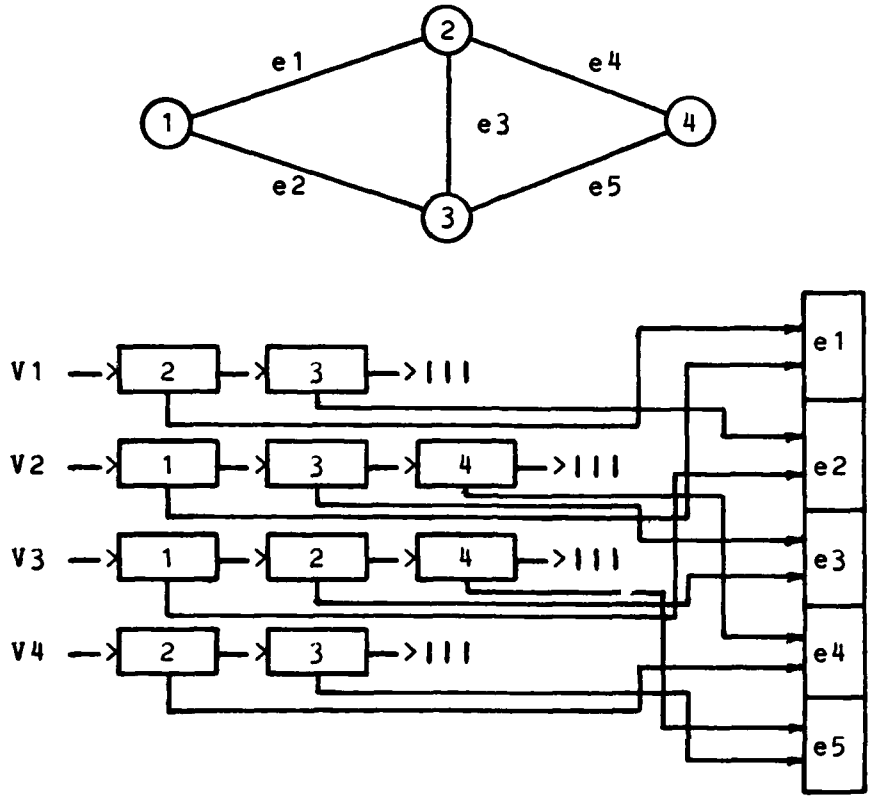


Figure I - Multilist Data Structure

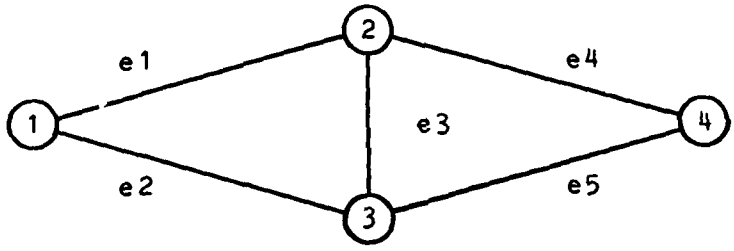
3.1.2.1. Data Structure Implementation

Next, we describe the FORTRAN arrays used to implement the code's data structures.

PTRADJ(i) points to the beginning of the list of vertices adjacent to vertex i. If PTRADJ(i) is positive, vertex i does not belong in set K. If PTRADJ(i) is negative, then vertex i is in set K. PTRADJ(i) may also be null. This indicates that vertex i has been removed from the network. Array ADJVRT(*) is the principal information element in the list structure. It contains an adjacent vertex to the vertex represented by the list in which it resides. If ADJVRT(*) is positive, that vertex is not in K, and if it is negative, the vertex is in set K. LNKDWN(i) points to the next element (downwards) in the list of adjacent vertices. If LNKDWN(*) = 0, this element is the last element in the list. LNKUP(*) points to the element just above it in the adjacent vertices list. If LNKUP(*) = 0, then this element is the first in the list. EDGPRB(*) contains the edge reliability. LNKEDG(*) points to the position in array

EDGPRB(*) corresponding to edge [PTRADJ(i),ADJVRT(*)]. AVSADJ is a pointer to the beginning of the list of available space [B75].

The algorithm also requires data structures for both the T list and the chain. Since the order of the vertices in both the T list and the chain is irrelevant to the code, linear stacks are used to represent both structures. TLIST(*) contains the stack of vertices, which are elements of the T list. If TLIST(*) is positive, then that vertex is not in set K. If it is negative, the vertex is in set K. TTOP points to the top of the TLIST(*) stack. CHAIN(*) is a stack of vertices belonging to a given chain. TOP points to its top. As with TLIST(*), if CHAIN(*) is positive, then that vertex is not in set K. If it is negative, the vertex is in set K. ONLIST(i) = .TRUE. implies that vertex i is on the T list, and ONLIST(i) = .FALSE. implies the contrary. DEG(i) indicates the degree of vertex i. Figure II illustrates the use of some of these arrays in network representation.



p(e1)=.5
p(e2)=.2
p(e3)=.3
p(e4)=.9
p(e5)=.8

	1	2	3	4
ONLIST(*)	F	T	T	F
DEG(*)	2	3	3	2
PTRADJ(*)	1	3	6	9

	ADJVRT(*)	LNKDWN(*)	LNKUP(*)	LNKEDG(*)
1	2	2	0	1
2	3	0	1	2
3	1	4	0	1
4	3	5	3	3
5	4	0	4	4
6	1	7	0	2
7	2	8	6	3
8	4	0	7	5
9	2	10	0	4
10	3	0	9	5

	1	2	3	4	5
EDGPRB(*)	.5	.2	.3	.9	.8

Figure II - FORTRAN Implementation of List Structure

3.2. Input Data Consistency Tests

As described later, the input of a vertex in set K will require that its vertex number come preceded by a negative sign. PolyChain tests consistency of the signs of the vertices, and in this manner will cease execution if there is any inconsistency in the sign of a vertex number. Edge reliability is also tested for values less than zero or greater than one.

3.3. Data Dictionary

Next, the code's variables are listed, with a brief description of each one.

ADJVRT(*)	vertex adjacent to vertex whose list it is on
ALFA	reduction parameter
AUX	auxiliary variable
AUXPTR	auxiliary pointer
AUXTOP	pointer to next to the last element of a chain
AVSADJ	pointer to beginning of list of available space
BETA	reduction parameter
BOTTOM	pointer to bottom of list
CARDE	cardinality of set E
CARDK	cardinality of set K
CARDL	size of list after input
CARDV	cardinality of set V
CCHAIN	absolute value of CHAIN
CHAIN(*)	chain stack
CTEST	consistency test variable
D1	degree of vertex V1
D2	degree of vertex V2
DATE	character string containing date
DEG(*)	degree of vertex
DELTA	reduction parameter
DENDEN	denominator in network density computation
DENSTY	input network density
ECHOIN	indicator of input echo
ECHOUT	indicator of reduced network output
EDGOUT	edge selected to leave network
EDGPRB(*)	edge reliability
EGOUTB	selected edge to be deleted from network
EGOUTC	selected edge to be deleted from network
ERRO	indicator of error in some routine
FIRST	key indicating first passage in output routine
FOUND	indicator of whether or not a polygon was found
HOUR	character string containing hour
GAMMA	reduction parameter
IN	value of FORTRAN input file
INON	key indicating input echo option
IOUT	value of FORTRAN output file
IPTR	pointer
IPTR1	pointer

IPTR2	pointer
IPTRK	pointer to vertex in set K
IPTRNK	pointer to vertex not in set K
LINECT	output line counter
LNKC	pointer
LNKDWN(*)	pointer to next element on list
LNKEDG(*)	pointer to corresponding edge
LNKUP(*)	pointer to element above in list
M	product of all omegas, see [W82]
MAXCHN	maximum number of elements in chain stack
MAXEDG	maximum number of edges
MAXLST	maximum number of elements in adjacent vertices list
MAXVRT	maximum number of vertices
MCARDE	cardinality of set E at start of procedure
MCARDK	cardinality of set K at start of procedure
MCARDV	cardinality of set V at start of procedure
MXSTKT	maximum number of elements in T list stack
NEWEDG	pointer to new edge
NR1	counter of reductions of type 1
NR2	counter of reductions of type 2
NR3	counter of reductions of type 3
NR4	counter of reductions of type 4
NR5	counter of reductions of type 5
NR6	counter of reductions of type 6
NR7	counter of reductions of type 7
NR8	counter of reductions of type 8
NSR	counter of series reductions
ND2R	counter of degree 2 reductions
NUMDEN	numerator in network density computation
OMEGA	reduction parameter
ONLIST(*)	indicator of presence in T list
OUTON	key indicating reduced graph output on report
OUTPUT	final solution
PA	reliability of edge a
PAGE	output page counter
PB	reliability of edge b
PC	reliability of edge c
PD	reliability of edge d
PE	reliability of edge e
PF	reliability of edge f
POINT	pointer
PR	updated edge reliability
PROB	reliability of edge (V1,V2)
PS	updated edge reliability
PT	updated edge reliability
PT13	pointer
PT23	pointer
PTR	pointer
PTRADJ(*)	pointer to beginning of list of adjacent vertices
PTRB	pointer
PTRC	pointer
PTRD2	pointer to VD2
PTRDG2	pointer to VDG2
PTRK	pointer to vertex in set K

PTRNK	pointer to vertex not in set K
PTRV	pointer
PTRV1	pointer to vertex adjacent to V
PTRV2	pointer to vertex adjacent to V
PV1	pointer
PV2	pointer
PTVK	pointer
PTVNK	pointer
QA	failure probability of edge a
QB	failure probability of edge b
QC	failure probability of edge c
QD	failure probability of edge d
QE	failure probability of edge e
QF	failure probability of edge f
TEST	key for debugging feature
TIME(*)	execution time
TLIST(*)	T list stack
TOP	top of chain stack
TTOP	top of TLIST stack
TYPE1	vertex type indicator for output
TYPE2	vertex type indicator for output
USEDGE	core usage for edge
USVRTX	core usage for vertex
USLIST	core usage for lists
V	vertex
V1	vertex
V2	vertex
V3	vertex adjacent to V2
V3V1	vertex adjacent to V1
V3V2	vertex adjacent to V2
V4	vertex
V5V1	vertex adjacent to vertex adjacent to V1
V5V2	vertex adjacent to vertex adjacent to V2
VD2	vertex with degree 2
VDG2	vertex with degree greater than 2
VK	vertex in set K
VNK	vertex not in set K
VRTX	vertex
VV	absolute value of V
VV1	absolute value of V1
VV2	absolute value of V2
VV3	absolute value of V3
VV3V1	absolute value of V3V1
VVDG2	absolute value of VDG2
VVNK	absolute value of VNK
W	vertex
X	vertex
XX	absolute value of X
Y	vertex
YY	absolute value of Y
YEAR	character string containing year

3.4. COMMON Blocks

Most of the variables, data structures, and parameters in PolyChain are shared by the subroutines through COMMON blocks. All of the dimensions of these COMMONs are determined in SUBROUTINE INILST, through the initialization of variables MAXEDG, MAXVRT, and MAXCHN. We will use these values when describing the blocks. Below is a list of all COMMON blocks in the code.

```
COMMON/BLK01/ DEG(MAXVRT)
COMMON/BLK02/ PTRADJ(MAXVRT),ADJVRT(2*MAXEDG),AVSADJ
COMMON/BLK21/ LNKDWN(2*MAXEDG),LNKUP(2*MAXEDG),LNKEDG(2*MAXEDG)
COMMON/BLK03/ EDGPRB(MAXEDG),EDGNUM(MAXEDG)
COMMON/BLK04/ TLIST(MAXVRT),CHAIN(MAXCHN),TTOP,TOP,
              ONLIST(MAXVRT)
COMMON/BLK05/ MAXEDG,MAXVRT,MAXLST,MXSTKT,MAXCHN
COMMON/BLK06/ CARDE,CARDV,CARDK
COMMON/BLK07/ M
COMMON/BLK08/ IN,IOUT
COMMON/BLK09/ FOUND
COMMON/BLK10/ IPTR1,IPTR2,IPTR,V1,V2,D1,D2
COMMON/BLK11/ PTRV1,PTRV2,PTRV
COMMON/BLK12/ V3V1,V3V2,V3,V4,V5V1,V5V2
COMMON/BLK14/ PT13,PT23
COMMON/BLK15/ P315
COMMON/BLK30/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NSR,ND2R
COMMON/BLK31/ MCARDE,MCARDV,MCARDK
COMMON/BLK32/ DATE,YEAR,HOUR
COMMON/BLK33/ FIRST
COMMON/BLK34/ TIME(2),PAGE
COMMON/BLK35/ INON,OUTON
```

3.5. Description of Subroutines

Next, we present all SUBROUTINES in the code. Each procedure will be briefly described together with its input and output.

3.5.1. MAIN Program

Description	The MAIN program controls the basic steps in the algorithm. From it, subroutines, representing the various steps of the algorithm, are called. The algorithm initiates and terminates execution in the MAIN program.
Input	An undirected network and edge reliabilities.
Output	Network reliability, when network is completely reduced. Reduced network and M, when total reduction is not possible.

3.5.2. SUBROUTINE INILST

Description — This subroutine initializes the code's most important variables. It creates the list of available space [B75], which is the first step in the initialization of the list structures.

Input — Variables and arrays not yet initialized.

Output — Initialized variables and data structures. The list of available space is also returned, ready to be loaded in SUBROUTINE INDATA(V).

3.5.3. SUBROUTINE INDATA(V)

Description — This subroutine reads the input network. Output options are read first. Each edge is read on a separate line. Edges are indicated by their corresponding vertices and reliability. A vertex number is preceded by a minus sign if the vertex is in set K. Consistency tests are performed on the input data, and if any inconsistency is reported, execution is terminated. Network multilist structure is loaded. Cardinalities and degrees are computed. Finally, the input network is optionally printed.

Input — Free format input. Output options. Network and edge reliabilities. No flags are needed to indicate end-of-file.

Output — The loaded multilist data structure. Vertex degrees. The input network is printed as a report if it is so desired.

3.5.4. SUBROUTINE HEADER

Description — This subroutine prints the page header of the input network output.

Input — Page number, date, year, and hour.

Output — Page header with incremented page number, and date, year, and hour of execution.

3.5.5. SUBROUTINE CONLST

Description — This subroutine constructs the initial T list.

Input — Degrees of all vertices.

Output — A stack, TLIST(*), containing the vertices on the T list. A pointer TTOP indicating the top of stack TLIST(*). An array, ONLIST(*),

indicating if a vertex is on the T list.

3.5.6. SUBROUTINE SERIER(V)

Description — This subroutine performs a series reduction on vertex V not in set K.

Input — Vertex V and the multilist structure.

Output — The updated multilist structure, with V and both of its edges deleted, and with a new edge inserted. This new edge has its reliability computed. New cardinalities of V and E.

3.5.7. SUBROUTINE DEG2R(V)

Description — This subroutine performs a degree 2 reduction on vertex V in set K.

Input — Vertex V and the multilist structure.

Output — The updated multilist structure, with V deleted, along with both of its edges, and with a new edge inserted. This new edge has its reliability computed. New cardinalities of V and E. The updated value of M.

3.5.8. SUBROUTINE SEARCH(V)

Description — This subroutine controls the process of searching for a polygon emanating from vertex V, and, if one is found, reducing it to a chain.

Input — Vertex V and the multilist structure.

Output — The updated multilist structure, with a polygon-to-chain reduction performed. A chain contained in stack CHAIN(*). Updated cardinalities and degrees. Updated M.

3.5.9. SUBROUTINE TYPE8(V)

Description — This subroutine searches for a polygon of type 8 (two parallel edges), emanating from vertex V. If found, a reduction is performed.

Input — Vertex V. The multilist structure.

Output — The updated multilist structure. A chain consisting of two vertices and one edge. An indicator (FOUND) is activated if a polygon was found.

3.5.10. SUBROUTINE DELETE(V,PTR)

Description — This subroutine deletes the element pointed to by PTR from vertex V's adjacent vertices list. Three cases are considered. The first, when the element is first in the list. The second, when it is last in the list. The last, when the element is in the middle of the list. In each case, the element is deleted by a different set of of commands.

Input — The multilist structure. Vertex V. Pointer PTR.

Output — The updated multilist structure without the specified element.

3.5.11. SUBROUTINE FINDK(V)

Description — This subroutine is called from control routine SEARCH(V), when vertex V is in set K. It controls the search for a polygon and, when one is found, its reduction to a chain. Only polygons of types 2 or 3 may be found emanating from a vertex V in set K.

Input — The multilist structure. Vertex V.

Output — This routine outputs pairs of adjacent vertices to vertex V, along with their degrees, to more specialized search routines (POLY2(V) and POLY3(V)). It also outputs the corresponding pointers to these vertices in vertex V's list. If a polygon is found, the indicator FOUND is returned "on" to subroutine SEARCH(V).

3.5.12. SUBROUTINE POLY2(V)

Description — This routine checks if vertex V, and the pair of adjacent vertices specified in subroutine FINDK(V) form a polygon of type 2. If a polygon of type 2 is found, this routine calls routine REDUC2(V), which performs the polygon-to-chain reduction.

Input — The multilist structure. Vertex V and a pair of vertices adjacent to V, along with their corresponding degrees. Pointers to the pair of adjacent vertices, in V's list.

Output — Indicator FOUND is activated if a polygon of type 2 is found.

3.5.13. SUBROUTINE REDUC2(V)

Description — This routine is called from subroutine POLY2(V), when a polygon of type 2 is found. It reduces the polygon found to a chain of three vertices.

Input — The multilist structure. The vertices of the polygon, and their pointers.

Output — The updated multilist structure. Updated degrees and cardinalities. A chain in stack CHAIN(*). Updated value of M.

3.5.14. SUBROUTINE POLY3(V)

Description — This routine checks if vertex V, and the pair of adjacent vertices specified in subroutine FINDK(V) form a polygon of type 3. If a polygon of type 3 is found, this routine calls routine REDUC3(V), which performs the polygon-to-chain reduction.

Input — The multilist structure. Vertex V and a pair of vertices adjacent to V, along with their corresponding degrees. Pointers to the pair of adjacent vertices, in V's list.

Output — Indicator FOUND is activated if a polygon of type 3 is found.

3.5.15. SUBROUTINE REDUC3(V)

Description — This routine is called from subroutine POLY3(V), when a polygon of type 3 is found. It reduces the polygon found to a chain of three vertices.

Input — The multilist structure. The vertices of the polygon, and their pointers.

Output — The updated multilist structure. Updated degrees and cardinalities. A chain in stack CHAIN(*). Updated value of M.

3.5.16. SUBROUTINE FINDNK(V)

Description — This subroutine is called from control routine SEARCH(V), when vertex V is not in set K. It controls the search for a polygon and its reduction to a chain, when a polygon is found. If vertex V is not in set K, any type of polygon can emanate from it.

Input — The multilist structure. Vertex V.

Output — This routine outputs to more specialized search routines [POLY2P(V), PY3467(V), and POLY15(V)] pairs of adjacent vertices of vertex V, and their degrees. It also outputs the corresponding pointers to these vertices in vertex V's list. If a polygon is found, the indicator FOUND is returned "on" to subroutine SEARCH.

3.5.17. SUBROUTINE PY3467(V)

Description — This routine checks if vertex V, and the pair of adjacent vertices specified in subroutine FINDNK(V) form a polygon of type 3,4,6, or 7. If a polygon of type 3,4,6, or 7 is found, this routine calls routine REDC3P(V), REDUC4(V), REDUC6(V), or REDUC7(V), respectively, which performs the polygon-to-chain reduction.

Input — The multilist structure. Vertex V and a pair of vertices adjacent to V. Pointers to the pair of adjacent vertices, in V's list.

Output — Indicator FOUND is activated if a polygon of type 3,4,6, or 7 is found. All vertices in the polygon found, together with pointers indicating their addresses, are passed to the polygon-to-chain reduction routines.

3.5.18. SUBROUTINE REDC3P(V)

Description — This routine is called from subroutine PY3467(V), when a polygon of type 3 is found. It reduces the polygon found to a chain of three vertices.

Input — The multilist structure. The vertices of the polygon, and their pointers.

Output — The updated multilist structure. Updated degrees and cardinalities. A chain in stack CHAIN(*). Updated value of M.

3.5.19. SUBROUTINE REDUC4(V)

Description — This routine is called from subroutine PY3467(V), when a polygon of type 4 is found. It reduces the polygon found to a chain of four vertices.

Input — The multilist structure. The vertices of the polygon, and their pointers.

Output — The updated multilist structure. Updated degrees and cardinalities. A chain in stack CHAIN(*). Updated value of M.

3.5.20. SUBROUTINE REDUC6(V)

Description — This routine is called from subroutine PY3467(V), when a polygon of type 6 is found. It reduces the polygon found to a chain of four vertices.

Input — The multilist structure. The vertices of the polygon, and their pointers.

Output — The updated multilist structure. Updated degrees and cardinalities. A chain in stack CHAIN(*). Updated value of M.

3.5.21. SUBROUTINE REDUC7(V)

Description — This routine is called from subroutine PY3467(V), when a polygon of type 7 is found. It reduces the polygon found to a chain of four vertices.

Input — The multilist structure. The vertices of the polygon, and their pointers.

Output — The updated multilist structure. Updated degrees and cardinalities. A chain in stack CHAIN(*). Updated value of M.

3.5.22. SUBROUTINE POLY2P(V)

Description — This routine checks if vertex V, and the pair of adjacent vertices specified in subroutine FINDNK(V) form a polygon of type 2, when V is not in set K. If a polygon of type 2 is found, this routine calls routine REDC2P(V), which performs the polygon-to-chain reduction.

Input — The multilist structure. Vertex V and a pair of vertices adjacent to V, along with their corresponding degrees. Pointers to the pair of adjacent vertices, in V's list.

Output — Indicator FOUND is activated if a polygon of type 2 is found.

3.5.23. SUBROUTINE REDC2P(V)

Description — This routine is called from subroutine POLY2P(V), when a polygon of type 2 is found emanating from vertex V, when V is not in set K. It reduces the polygon found to a chain of three vertices.

Input — The multilist structure. The vertices of the polygon, and their pointers.

Output — The updated multilist structure. Updated degrees and cardinalities. A chain in stack CHAIN(*). Updated value of M.

3.5.24. SUBROUTINE POLY15(V)

Description — This routine checks if vertex V, and the pair of adjacent vertices specified in subroutine FINDNK(V) form a polygon of type 1 or 5. If a polygon of type 1 or 5 is found, this routine calls routine REDUC1(V) or REDUC5(V), respectively, which performs the polygon-to-chain reduction.

Input — The multilist structure. Vertex V and a pair of vertices adjacent to V. Pointers to the pair of adjacent vertices, in V's list.

Output — Indicator FOUND is activated if a polygon of type 1 or 5 is found. All vertices in the polygon found, together with pointers indicating their addresses, are passed to the polygon-to-chain reduction routines.

3.5.25. SUBROUTINE REDUC1(V)

Description — This routine is called from subroutine POLY15(V), when a polygon of type 1 is found. It reduces the polygon found to a chain of three vertices.

Input — The multilist structure. The vertices of the polygon, and their pointers.

Output — The updated multilist structure. Updated degrees and cardinalities. A chain in stack CHAIN(*). Updated value of M.

3.5.26. SUBROUTINE REDUC5(V)

Description — This routine is called from subroutine POLY15(V), when a polygon of type 5 is found. It reduces the polygon found to a chain of two

or four vertices, depending on the cardinality of set K.

Input — The multilist structure. The vertices of the polygon, and their pointers.

Output — The updated multilist structure. Updated degrees and cardinalities. A chain in stack CHAIN(*). Updated value of M.

3.5.27. SUBROUTINE CHNRED

Description — This routine performs a complete reduction on the chain resulting from a polygon-to-chain reduction, when this reduction has not led to a cycle. Vertices V and W are the chains end-points. The routine analyzes three cases: a) deg(V) > 2 and deg(W) = 2; b) deg(V) = 2 and deg(W) > 2; and c) deg(V) > 2 and deg(W) > 2. When needed, series and degree 2 reductions are performed and the T list is updated.

Input — The multilist structure. The chain in th CHAIN(*) array. The degrees of the vertices. The ONLIST(*) array. The T list.

Output — The updated multilist structure. The updated T list.

3.5.28. SUBROUTINE OUTGRF

Description — This routine optionally prints out the network when it has not been totally reduced to a two-vertices-one-edge graph. A data file is always written containing M and the reduced network.

Input — The multilist structure. The current value of M.

Output — A report is generated, if so desired, containing the edges of the reduced network, along with each edge reliability. In this report, the current value of M is also listed. A data file with M and the reduced network.

3.5.29. SUBROUTINE HEAD1

Description — This subroutine prints the page header of the reduced network output.

Input — Page number, date, year, and hour.

Output — Page header with incremented page number, and date, year, and hour of execution.

3.5.30. SUBROUTINE OUTREL

Description — This routine prints out the network when it has been totally reduced to a two-vertices-one-edge graph.

Input — The multilist structure. The current value of M.

Output — A report is generated, containing the network's reliability.

3.5.31. SUBROUTINE HEAD2

Description — This subroutine prints the page header of the intermediate and final results output.

Input — Page number, date, year, and hour.

Output — Page header with incremented page number, and date, year, and hour of execution.

3.6. Debugging Feature for Program Maintenance

A desirable feature in any code, is the possibility of having intermediate printouts of important variables and arrays, so that maintenance can be simplified. In this code, most SUBROUTINEs have a LOGICAL variable named TEST which, when "turned on", i.e. when made equal to .TRUE., will activate such a debugging feature.

4. User's Guide

In this section of the report, a user's manual is presented. First, we present a guide for using PolyChain in an application. The input file and outputs are then described. Finally, two test problems are run on the code to illustrate outputs.

4.1. Executing PolyChain

Here, we comment on how the program is executed. The algorithm requires that the network be nonseparable. An algorithm that tests for this condition using Depth First Search is found in [A74]. To run the code, first the dimension parameters and the COMMON blocks must be adjusted. Then, an input data file must be prepared. Both of these topics are presented below.

4.1.1. Dimension Parameters

The first step in running PolyChain, is adjusting the dimension parameters SUBROUTINE INILST.

These parameters are: MAXVRT - maximum number of vertices
MAXEDG - maximum number of edges

If the dimension of the problem being run is smaller than the dimensions already specified in this subroutine, no adjustment is needed. But if the network's dimensions exceed what has been specified, the values of MAXEDG and MAXVRT must be altered. Note that parameter MAXCHN need not be modified.

After adjusting the dimension parameters, all COMMON blocks containing arrays must be changed accordingly. These COMMON blocks are adjusted with the dimensions specified below.

```
COMMON/BLK01/ DEG(MAXVRT)
COMMON/BLK02/ PTRADJ(MAXVRT),ADJVRT(2*MAXEDG),AVSADJ
COMMON/BLK21/ LNKDWN(2*MAXEDG),LNKUP(2*MAXEDG),LNKEDG(2*MAXEDG)
COMMON/BLK03/ EDGPRB(MAXEDG),EDGNUM(MAXEDG)
COMMON/BLK04/ TLIST(MAXVRT),CHAIN(MAXCHN),TTOP,TOP,
              ONLIST(MAXVRT)
```

4.1.2. Input Files

Inputing data is very simple in PolyChain. All input is format free, i.e., data is not restricted to specific columns of the input line. No flag is needed to indicate end-of-file.

The first line of the input file contains the system output options. There are two entries on this line: ECHOIN and ECHOUT. Set:

ECHOIN = { 1 - if a report of the input network is desired
         { 0 - otherwise

ECHOUT = { 1 - if a report of the reduced series-parallel complex network is desired
         { 0 - otherwise

Setting ECHOUT to "0" will not hinder PolyChain from writing a data file with M and the reduced network, when the input network is series-parallel complex.

The edges are entered one by one on each line. To specify an edge, enter both vertices of the edge followed by the edge's reliability. If a vertex is distinguished, that is, belongs to set K, it should be preceded by a minus sign. Below, we illustrate an input file, for the network in figure III.

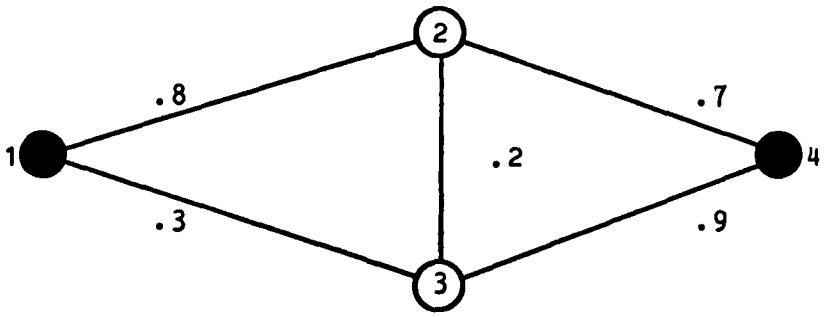


Figure III - Network for Example Input File

For this network, the input file is given below. Here we wish no input network report, but require a report of the reduced series-parallel complex network.

```
 0    1
-1    2   .8
-1    3   .3
 2    3   .2
```

```
 2   -4   .7
 3   -4   .9
```

4.1.3. Program Outputs

In this section PolyChain is tested with two different test problems illustrating the program's outputs.

4.1.3.1. Problem One

Consider a modification of the ARPA Computer Network in figure IV.

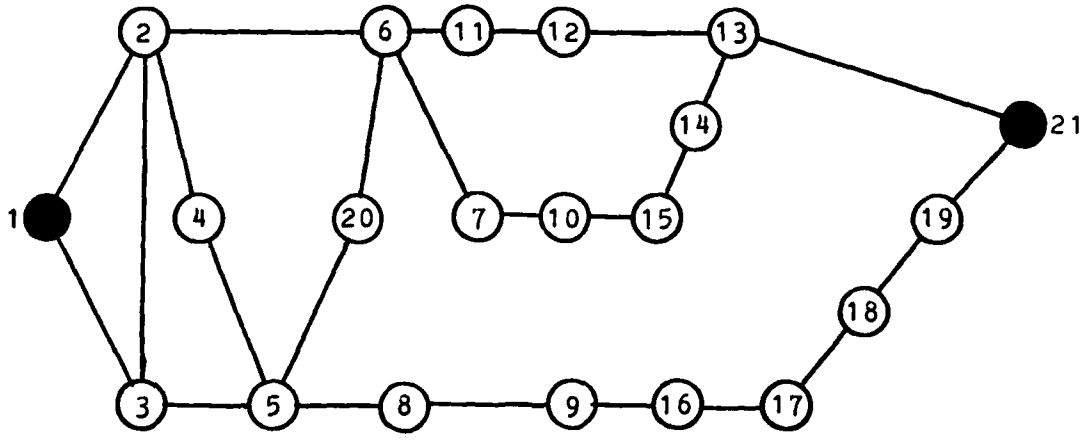


Figure IV - Modified ARPA Computer Network

Consider edge reliabilities for this network varying between 0.1 and 0.9. Both output options are set to "1". The input file for this network is given below.

```
 1  1
-1  2  .1
-1  3  .2
 2  3  .3
 2  4  .4
 2  6  .5
 3  5  .6
 4  5  .7
 5  8  .8
 5 20  .9
 6 11  .1
 6 20  .2
```

```
 6  7  .3
 7 10  .4
 8  9  .5
10 15  .7
11 12  .8
12 13  .9
13 14  .1
13 -21 .2
14 15  .3
 9 16  .4
16 17  .5
17 18  .6
18 19  .7
19 -21 .8
```

The modified ARPA network is series-parallel reducible. PolyChain was run with the above input data. For this type of network the code generates a two part report. The first part describes the input network, edge by edge, with edge numbers, vertex numbers and types (K=K-vertex, nK=non K-vertex), and edge reliability. This section also summarizes the input network data and core usage. The second part of the report indicates that the network is series-parallel reducible, contains the K-terminal network reliability, summarizes the reductions performed by the algorithm and indicates the CPU time (excluding I/O). The three page report generated by PolyChain for the above file follows.

PolyChain - Version 83.1
Polygon to Chain Reductions
in Network Reliability

Date : Tue Jul 26 1983
Time : 12:59:52

Input Network

Edge	Vertex	Type	Vertex	Type	Reliability
1	1	K	2	nK	.10000000d+00
2	1	K	3	nK	.20000000d+00
3	2	nK	3	nK	.30000000d+00
4	2	nK	4	nK	.40000000d+00
5	2	nK	6	nK	.50000000d+00
6	3	nK	5	nK	.60000000d+00
7	4	nK	5	nK	.70000000d+00
8	5	nK	8	nK	.80000000d+00
9	5	nK	20	nK	.90000000d+00
10	6	nK	7	nK	.10000000d+00
11	6	nK	11	nK	.20000000d+00
12	6	nK	20	nK	.30000000d+00
13	7	nK	10	nK	.40000000d+00
14	8	nK	9	nK	.50000000d+00
15	9	nK	16	nK	.70000000d+00
16	10	nK	15	nK	.80000000d+00
17	11	nK	12	nK	.90000000d+00
18	12	nK	13	nK	.10000000d+00
19	13	nK	14	nK	.20000000d+00
20	13	nK	21	K	.30000000d+00
21	14	nK	15	nK	.40000000d+00

PolyChain - Version 83.1 Page 2
Polygon to Chain Reductions
in Network Reliability

Date : Tue Jul 26 1983
Time : 12:59:52

Input Network

Edge	Vertex	Type	Vertex	Type	Reliability
22	16	nK	17	nK	.50000000d+00
23	17	nK	18	nK	.60000000d+00
24	18	nK	19	nK	.70000000d+00
25	19	nK	21	K	.80000000d+00

Summary of Input Network Data

Number of Vertices..... 21
Number of Edges..... 25
Number of K-Vertices..... 2
Network Density..... 0.119

Summary of Core Usage

Variable Name	Current Value	Usage	%
MAXEDG	5000	25	0.5
MAXVRT	2000	21	1.0

PolyChain - Version 83.1
Polygon to Chain Reductions
in Network Reliability

Date : Tue Jul 26 1983
Time : 12:59:52

Network Series-Parallel Reducible

Network Reliability..... 0.87028600d-02

Reductions Performed

Series.....	19
Degree 2.....	0
Type 1.....	3
Type 2.....	0
Type 3.....	0
Type 4.....	0
Type 5.....	0
Type 6.....	0
Type 7.....	0
Type 8.....	1

Solution Time = 0.02 Secs.

4.1.3.2. Problem Two

We now consider a series-parallel complex network, the ARPA computer network, in figure V.

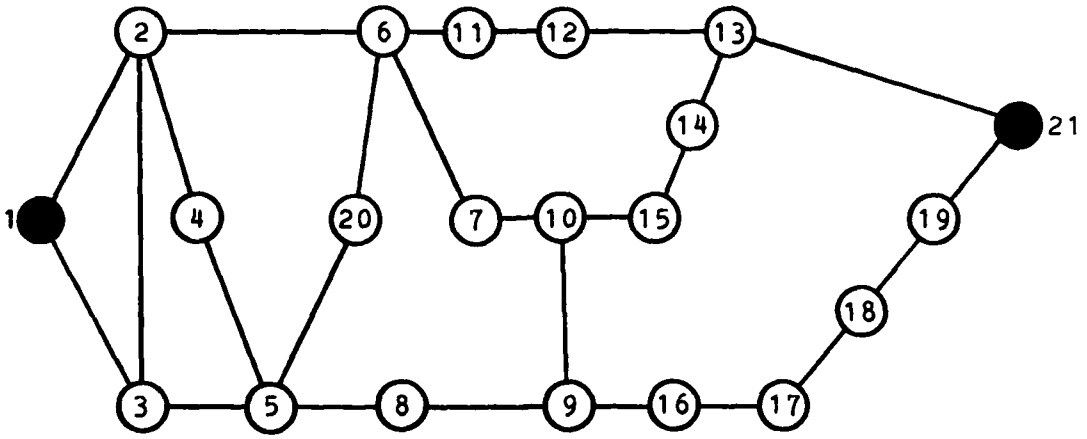


Figure V - ARPA Computer Network

This network's input file is given next. Edge reliabilities vary from 0.1 to 0.9, while output options are both set to "1".

```
  1   1
 -1   2   .1
 -1   3   .2
  2   3   .3
  2   4   .4
  2   6   .5
  3   5   .6
  4   5   .7
  5   8   .8
  5  20   .9
  6  11   .1
  6  20   .2
  6   7   .3
  7  10   .4
  8   9   .5
  9  10   .6
 10  15   .7
 11  12   .8
 12  13   .9
 13  14   .1
 13 -21   .2
 14  15   .3
```

```
 9 16  .4
16 17  .5
17 18  .6
18 19  .7
19 -21 .8
```

For series-parallel complex networks, PolyChain also generates a two part report together with an output file. The first section of the report is identical to the series-parallel reducible case, that is, contains the input network. The second part indicates that the network is series-parallel complex, contains the reduced network, the updated value of M, and summarizes the reductions performed, including percentage reductions in the network's dimensions. CPU time (excluding I/O) is also indicated. The four page report generated by the code for the ARPA computer network is given next.

PolyChain - Version 83.1 Page 1
Polygon to Chain Reductions
in Network Reliability

Date : Tue Jul 26 1983
Time : 12:58:26

Input Network

Edge	Vertex	Type	Vertex	Type	Reliability
1	1	K	2	nK	.10000000d+00
2	1	K	3	nK	.20000000d+00
3	2	nK	3	nK	.30000000d+00
4	2	nK	4	nK	.40000000d+00
5	2	nK	6	nK	.50000000d+00
6	3	nK	5	nK	.60000000d+00
7	4	nK	5	nK	.70000000d+00
8	5	nK	8	nK	.80000000d+00
9	5	nK	20	nK	.90000000d+00
10	6	nK	7	nK	.10000000d+00
11	6	nK	11	nK	.20000000d+00
12	6	nK	20	nK	.30000000d+00
13	7	nK	10	nK	.40000000d+00
14	8	nK	9	nK	.50000000d+00
15	9	nK	10	nK	.60000000d+00
16	9	nK	16	nK	.70000000d+00
17	10	nK	15	nK	.80000000d+00
18	11	nK	12	nK	.90000000d+00
19	12	nK	13	nK	.10000000d+00
20	13	nK	14	nK	.20000000d+00
21	13	nK	21	K	.30000000d+00

PolyChain - Version 83.1
Polygon to Chain Reductions
in Network Reliability

Date : Tue Jul 26 1983
Time : 12:58:26

Input Network

Edge	Vertex	Type	Vertex	Type	Reliability
22	14	nK	15	nK	.40000000d+00
23	16	nK	17	nK	.50000000d+00
24	17	nK	18	nK	.60000000d+00
25	18	nK	19	nK	.70000000d+00
26	19	nK	21	K	.80000000d+00

Summary of Input Network Data

Number of Vertices..... 21
Number of Edges..... 26
Number of K-Vertices..... 2
Network Density..... 0.124

Summary of Core Usage

Variable Name	Current Value	Usage	%
MAXEDG	5000	26	0.5
MAXVRT	2000	21	1.0

PolyChain - Version 83.1 Page 3
Polygon to Chain Reductions
in Network Reliability

Date : Tue Jul 26 1983
Time : 12:58:26

Network Series-Parallel Complex
Reduced Network

Edge	Vertex	Type	Vertex	Type	Reliability
14	1	K	9	nK	.31422812d+00
5	1	K	6	nK	.53817522d+00
19	6	nK	13	nK	.18000000d-01
13	6	nK	10	nK	.40000000d-01
26	9	nK	21	K	.11760000d+00
15	9	nK	10	nK	.60000000d+00
22	10	nK	13	nK	.64000000d-01
21	13	nK	21	K	.30000000d+00

PolyChain - Version 83.1
Polygon to Chain Reductions
in Network Reliability

Date : Tue Jul 26 1983
Time : 12:58:26

Updated value of M = 0.21679720d+00

Reductions Performed

Series.....	15
Degree 2.....	0
Type 1.....	3
Type 2.....	0
Type 3.....	0
Type 4.....	0
Type 5.....	0
Type 6.....	0
Type 7.....	0
Type 8.....	0

	Original Network	Reduced Network	% Reduction
Edges.....	26	8	69.2
Vertices.....	21	6	71.4
K-Vertices.....	2	2	0.

Solution Time = 0.02 Secs.

For the series-parallel complex case, PolyChain also writes a file with the updated value of M and the reduced network. We next present the file, named "polychain.out", generated in this example. This file can be used as input to a factoring algorithm program.

```
 0.21679720d+00
      -1       9   0.31422812d+00
      -1       6   0.53817522d+00
       6      13   0.18000000d-01
       6      10   0.40000000d-01
       9     -21   0.11760000d+00
       9      10   0.60000000d+00
      10      13   0.64000000d-01
      13     -21   0.30000000d+00
```

The reduced network obtained by PolyChain for the ARPA computer network is given in figure VI.

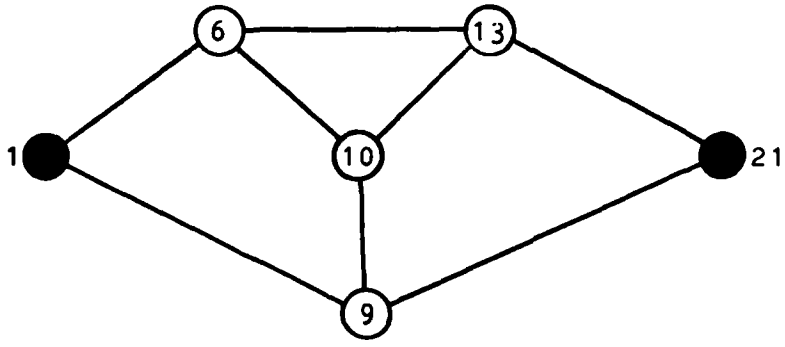


Figure VI - Reduced Network

5. Large-Scale Networks

Next, we test PolyChain with a number of large-scale networks. These networks help us understand PolyChain's behavior when treating large networks and enable the estimation of CPU times for even larger problems. A random network generator was used to provide the series-parallel complex networks, while the series-parallel reducible networks were built aggregating a number of 150 vertex, 208 edge networks. Problems were run on the VAX 11/750 of the Etcheverry Hall VAX/UNIX CAE Laboratory, at Berkeley. The code was compiled on the UNIX f77 compiler using the -O optimizing option. The code was run under the UNIX operating system. CPU times were measured through the "dtime" system routine. Table I contains a summary of the networks tested.

Prob	Edges	Vertices	K-Vertices	Density	Reducible ?
1	208	150	63	.019	Yes
2	418	300	126	.009	Yes
3	838	600	252	.005	Yes
4	1258	900	328	.003	Yes
5	2518	1800	756	.002	Yes
6	211	150	63	.019	No
7	428	300	126	.010	No
8	840	600	252	.005	No
9	1272	900	328	.003	No
10	2521	1800	756	.002	No

Table I - Large-Scale Test Problems

The results obtained for these problems are contained table II.

Prob	R E D U C T I O N S										Mean CPU Time
	Ser	Deg2	Ty1	Ty2	Ty3	Ty4	Ty5	Ty6	Ty7	Ty8	
1	87	56	8	24	3	0	5	0	1	17	0.27s
2	174	114	16	49	6	0	10	0	2	35	0.57s
3	348	230	32	99	12	0	20	0	4	71	1.09s
4	522	346	48	149	18	0	30	0	6	107	1.60s
5	1044	694	96	299	36	0	60	0	12	215	3.23s
6	52	9	0	1	0	1	0	0	0	3	0.30s
7	98	18	1	1	0	0	0	0	0	1	0.60s
8	216	50	0	2	0	0	0	0	0	4	1.14s
9	330	36	2	1	0	0	0	0	0	1	1.66s
10	583	101	0	2	1	0	0	0	0	1	3.58s

Table II - Test Results

Each test problem was run 10 times. The problems, however, do not cover a very wide spectrum of networks and therefore some bias may exist in our conclusions.

The tests agree with the classification of the algorithm as being $O(|E|)$. The network size ($|E|$) processed per CPU second varied from 733.33 to 784.28 edges/sec for the series-parallel reducible networks tested and from 694.08 to 766.26 edges/sec on the series-parallel complex networks. The mean network size ($|E|$) processed per CPU second was, respectively, 775.37 and 725.17 edges/sec for the reducible and complex networks.

These first results show that PolyChain is feasible for computing K-terminal reliability of large-scale series-parallel reducible networks. With 1 min. CPU, one should be able to compute the K-terminal reliability of a series-parallel reducible network upwards of 35,000 edges.

6. Conclusions and Recommendations

6.1. Conclusions

This report discussed the design and implementation of PolyChain, a portable FORTRAN code for evaluating undirected network reliability via polygon-to-chain reductions, [S82], [W82]. The code's implementation facilitates further extensions and enhancements. It uses a multilist data structure representation of the network, which enables good core management and efficient network manipulation. Input is simple and format free. Input data is tested for consistency and execution is terminated, when inconsistent data is detected. Outputs are detailed and a data file, containing the reduced network, is written, when the network is series-parallel complex.

The code was tested on large-scale networks and the results are encouraging.

A copy of the program's source code is available through the author.

6.2. Recommendations

Further testing is still needed to ensure the code's correctness. A random network generator is also required, so that large-scale networks can be further tested on PolyChain. This generator should have the capability of generating both series-parallel reducible and complex networks.

The algorithm has a constraint on the topology of the input network. The network must be nonseparable. In the present version, PolyChain does not test for this requirement. Since these networks may occur in practice, a routine to check for this condition is desirable. Aho, Hopcroft, and Ullman, [A74], present a Depth First Search based algorithm for determining all nonseparable components of a network.

To insure the evaluation of the K-terminal network reliability for any network, this code should encorporate a factoring algorithm. In this manner, the program could oscillate between both algorithms until a final result is attained.

7. References

[A74] Aho,A.V.,Hopcroft,J.E.,Ullman,J.D., The Design and Analysis of Computer Algorithms, pp. 185-186, Addison-Wesley Publishing Company, 1974

[A83] Agrawal,A.,Barlow,R.E., "A Survey of Network Reliability", ORC 83-5, Operations Research Center, University of California, Berkeley, 1983

[B75] Berztiss,A.T., Data Structures: Theory and Practice, Academic Press, 1975

[H80] Helgason,R.V.,Kennington,J.L., Algorithms for Network Programming, John Wiley and Sons, 1980

[K73] Knuth,D.E., The Art of Computer Programming, Volume One: Fundamental Algorithms, Addison-Wesley, 1973

[T78] Thesen,A., Computer Methods in Operations Research, Academic Press, 1978

[S82] Satyanarayana,A.,Wood,R.K., "Polygon-to-Chain Reductions and Network Reliability", ORC 82-4, Operations Research Center, University of California, Berkeley, 1982

[W82] Wood,R.K., "Polygon-to-Chain Reductions and Extensions for Reliability Evaluation of Undirected Networks", Ph.D. Thesis, Dept. of Industrial Engineering and Operations Research, University of California, Berkeley, 1982

P o l y C h a i n

A COMPUTER PROGRAM FOR RELIABILITY EVALUATION OF LARGE-SCALE UNDIRECTED NETWORKS VIA POLYGON-TO-CHAIN REDUCTIONS[†]

Operations Research Center Research Report No. 83-10

Mauricio G. C. Resende

October 1983

U. S. Army Research Office - Research Triangle Park

DAAG29-81-K-0160

Operations Research Center
University of California, Berkeley

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED.

[†]Partially supported by Conselho Nacional de Desenvolvimento Cientifico e Tecnologico - CNPq, Brazil.

THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF THE AUTHOR(S) AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.