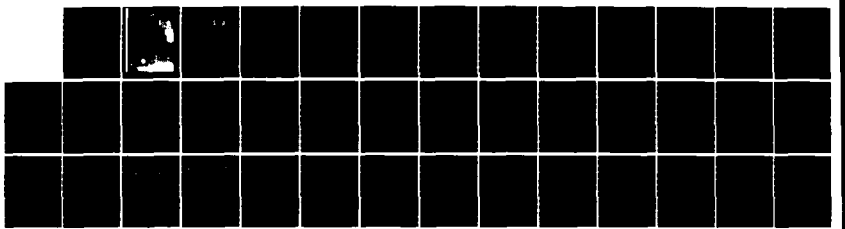
AD-A164 242 EXTENSIONS TO POLYCHAIN: NONSEPARABILITY TESTING AND FACTORING ALGORITHM(U) CALIFORNIA UNIV BERKELEY OPERATIONS RESEARCH CENTER L I RESENDE 02 DEC 85 1/1

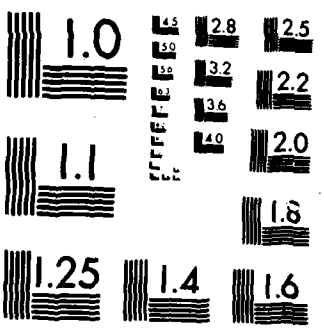
UNCLASSIFIED ORC-85-14 N00014-85-K-0384 F/G 9/2 NL



END

FILMED

DTIC



OPERATIONS RESEARCH CENTER



# EXTENSIONS TO POLYCHAIN: NONSEPARABILITY TESTING AND FACTORING ALGORITHM

by

Lucia I. P. Resende*

ORC 85-14 December 1985



UNIVERSITY OF CALIFO

(12)



# EXTENSIONS TO POLYCHAIN: NONSEPARABILITY TESTING AND FACTORING ALGORITHM

by

Lucia I. P. Resende*

ORC 85-14 December 1985

*Department of Industrial Engineering and Operations Research, University of California, Berkeley, California.

This research was supported by the Office of Naval Research under contract N00014-K-0384 with the University of California, and 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.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>ORC 85-14 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>Extensions to Polychain: Nonseparability Testing and Factoring Algorithm | | 5. TYPE OF REPORT & PERIOD COVERED<br>Technical<br>6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s)<br>Lucia I. P. Resende | | 8. CONTRACT OR GRANT NUMBER(s)<br>N00014-K-0384 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Operations Research Center<br>University of California<br>Berkeley, CA 94720 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>Office of Naval Research<br>Dept. of the Navy<br>Arlington, VA 22217 | | 12. REPORT DATE<br>December 1985<br>13. NUMBER OF PAGES<br>39 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | | 15. SECURITY CLASS. (of this report)<br>Unclassified<br>15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Program
Data Structures
Graph Reproductions
Network Reliability
Factoring Algorithm
Series-Parallel Graphs

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(See Abstract)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

# Extensions to PolyChain: Nonseparability Testing and Factoring Algorithm

*Lucia I. P. Resende*

Operations Research Center

University of California, Berkeley

*ABSTRACT*

This report discusses the design and implementation of FORTRAN subroutines to add the capabilities of nonseparability testing and pivotal decomposition to *PolyChain*, a program for reliability evaluation of undirected networks via polygon-to-chain reductions.

December 2, 1985

# Extensions to PolyChain: Nonseparability Testing and Factoring Algorithm

*Lucia I. P. Resende*

Operations Research Center

University of California, Berkeley

## 1. Introduction

*PolyChain* is a portable FORTRAN program for evaluating the reliability of a K-terminal network via polygon-to-chain reductions and the factoring algorithm. The first version of *PolyChain* allows the evaluation of the K-terminal reliability for series-parallel graphs [1]. The algorithm implemented is a linear time algorithm introduced in 1982 by Satyanarayana and Wood [2,3].

This report discusses the design and implementation of two features recommended in a previous document that enable *PolyChain* subroutines to treat a larger class of problems. The algorithm of Satyanarayana and Wood has a constraint on the topology of the input network, requiring it to be nonseparable. The original version of *PolyChain* does not test for this requirement. One of the features added to *PolyChain* discussed in this report is the implementation of a routine to check for this condition.

The algorithm of Satyanarayana and Wood computes the reliability of a network with an underlying series-parallel structure. When the input network is not totally reducible an extension to the algorithm is required to obtain the reduced network. The second feature discussed

in this report is the implementation of a factoring algorithm incorporated to *PolyChain* to insure the evaluation of the K-terminal network reliability for both series-parallel reducible and irreducible networks.

Section 2 briefly presents some theoretical results of polygon-to-chain reductions. In section 3, the algorithm implemented for nonseparability testing is presented. Section 4 briefly discusses the factoring algorithm. A system manual is presented in section 5 describing the implementation of both algorithms in FORTRAN, and a user manual is presented in section 6. The code's performance is illustrated in section 7 through the testing of several networks. Conclusions and recommendations are made in section 8.

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

In this section a brief discussion of series-parallel graphs and polygon-to-chain reductions is presented. For a complete discussion see [2,3].

Throughout this report we consider an undirected graph $G=(V,E)$, where $V$ is the set of vertices and $E$ the set of edges of $G$. A connected graph $G=(V,E)$ is said to be *separable* if there exists a vertex $v$, called the *separation vertex*, such that its removal from the graph disconnects the graph. When a graph has no separation vertex, it is called *nonseparable*. The induced nonseparable subgraphs of a separable graph G are called *nonseparable components* of G.

Let $G=(V,E)$ be a nonseparable graph. Vertices are assumed to be perfectly reliable, and edges may fail, independently of each other, with known probabilities. The edge reliability for edge $e_i$ is $p_i$, and the edge-failure probability is $q_i=1-p_i$. Let $K \subseteq V$, $|K| \geq 2$ be a specified set of vertices. Vertices in $K$ will be referred to as *$K$-vertices*. $G_K$ is graph $G$ with $K$ specified. The *$K$-terminal reliability* of $G_K$, $R(G_K)$, is the probability that all K-vertices in $G_K$ are connected by working edges.

The size of graph $G_K$, i.e. $|V(G_K)| + |E(G_K)|$, can be reduced by applying *reliability-preserving reductions*. The application of reliability-preserving reductions to $G_K$ renders a graph $G'_K$ such that $R(G_K) = \Omega R(G'_K)$, where $\Omega$ is a multiplicative factor that depends on the reductions applied.

Three types of reliability-preserving reductions will be referred to as *simple reductions*: parallel reduction, series reduction, and degree-2 reduction. In *parallel reduction*, parallel edges $e_a=(x,y)$ and $e_b=(x,y)$ are replaced by a single edge $e_c=(x,y)$ with edge probability $p_c=1-q_a q_b$. In *series reduction*, edges $e_a=(x,y)$ and $e_b=(y,z)$ are replaced by a single edge $e_c=(x,z)$ with edge probability $p_c=p_a p_b$. For both series and parallel reductions the multiplicative factor $\Omega$ has value 1, and $K'=K$. In *degree-2 reduction* edges $e_a=(x,y)$ and $e_b=(y,z)$, $x, y, z \in K$, are replaced by edge $e_c=(x,z)$ with $p_c=p_a p_b/(1-q_a q_b)$, $\Omega=1-q_a q_b$, and $K'=K-y$.

Replacing a pair of series (parallel) edges by a single edge is called a series (parallel) replacement. A *replacement*, as opposed to a reduction, does not involve probabilities or a set of distinguished nodes associated with the graph.

A *nonseparable series-parallel graph* is a graph that can be reduced to a single edge by successive series and parallel replacements. If the graph is separable, it is series-parallel if it can be reduced to a tree after all possible series and parallel replacements are performed. A nonseparable series-parallel graph $G_K$ is termed *s-p reducible* if it can be reduced to a single edge by successive simple reductions. A graph $G_K$ is *s-p irreducible* if it is not *s-p reducible*.

A *chain* is an alternating sequence of distinct vertices and edges, such that the internal vertices are all of degree 2 and end vertices are of degree greater than 2. A chain must contain at least one edge and two end vertices. A *polygon* is a cycle such that exactly two vertices of the cycle are of degree greater than 2.

A set of reliability-preserving reductions introduced by Satyanarayana and Wood [2,3], replaces a polygon with a chain. These reductions are called *polygon-to-chain reductions*. It is shown in [2,3], that every series-parallel graph is reducible, irrespective of the vertices chosen to be in K, with the use of simple reductions and polygon-to-chain reductions. Making use of these two types of reliability-preserving reductions, a linear time algorithm to evaluate $R(G_K)$ for a series-parallel graph with any chosen set $K$ is presented in [2,3].

*PolyChain* is a direct implementation of that algorithm utilizing an extension so that a reduced network can be obtained when the graph is s-p irreducible. When a reduced graph is generated the factoring algorithm is applied to find the reliability of the reduced network. This way, *Polychain* can evaluate the K-terminal reliability of general nonseparable networks.

### 3. Nonseparability Testing

A depth-first-search based algorithm, having time complexity O(|E|), exists to detect separating vertices. This algorithm is implemented in *PolyChain* and is presented below as described in [9].

Assume that $|V|>1$, and $s$ is the vertex in which we start the search.

(1) Mark all edges "unused". Empty the stack $S$.

For every $v \in V$ let $k(v) \leftarrow 0$. Let $i \leftarrow 0$ and $v \leftarrow s$.

(2) $i \leftarrow i+1$, $k(v) \leftarrow i$, $L(v) \leftarrow i$ and put $v$ on $S$.

(3) If $v$ has no unused incident edges go to Step (5).

(4) Choose an unused incident edge $e=(v,u)$. Mark $e$ "used".

If $k(u) \neq 0$, let $L(v) \leftarrow Min\ \{L(v), k(v)\}$ and go to Step (3).

Otherwise ($k(u) = 0$) let $f(u) \leftarrow v$, $v \leftarrow u$ and go to Step (2).

(5) If $k(f(v)) = 1$, go to Step (9).

(6) ($f(v) \neq s$). If $L(v) < k(f(v))$, then $L(f(v)) \leftarrow Min\ \{L(f(v)), L(v)\}$ and go to Step (8).

(7) ($L(v) \geq k(f(v))$) $f(v)$ is a *separating vertex*.

All the vertices on $S$ down to and including $v$ are now removed from $S$; this set, with $f(v)$, forms a *nonseparable component*.

(8) $v \leftarrow f(v)$ and go to Step (3).

(9) All vertices on $S$ down to and including $v$ are now removed from $S$; tney form with $s$ a *nonseparable component*.

(10) If $s$ has no unused incident edges then halt.

(11) Vertex $s$ is a *separating vertex*. Let $v \leftarrow s$ and go to Step (4).

## 4. Factoring Algorithm

As already mentioned, when the input network has no underlying series-parallel structure, the polygon-to-chain algorithm generates a reduced network but does not compute the network's K-terminal reliability. We will discuss the implementation of the factoring algorithm incorporated to *PolyChain* to calculate the reliability of the reduced network generated when the input network is s-p irreducible.

### 4.1. The Algorithm

The K-terminal reliability, $R_K(G)$, of a graph G can be computed by repeated applications of the following decomposition,

$$R_K(G)=p_e R(G_e)+(1-p_e)R(G_{-e})$$

where $G_e$ is the graph obtained from G by considering that edge $e$ is working and $G_{-e}$ is the graph obtained from G when edge $e$ is not working. Hence, $G_e$ and $G_{-e}$ are obtained by respectively contracting and deleting edge $e$ in G.

After each application of this decomposition, simple reductions are performed. If the generated subgraph is not totally reduced a new edge is then selected and the decomposition

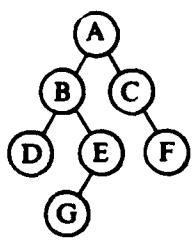
reapplied.

The following scheme describes, in recursive form, the factoring algorithm.

```
factor (G)
  reduce (G)
  select edge e to pivot
  factor (G_e)
  factor (G_-e)
end
```

The use of the factoring algorithm generates a binary computational tree whose root node is the original graph and each other node is a subgraph. Without the application of simple reductions after each edge selection, the binary structure would contain $2^{|E|}$ leaves, which is equivalent to the enumeration of all possible states of G. Notice that the scheme given above traverses the binary computational tree using a preorder enumeration. An example of a preorder traversal applied to a tree is given in figure 1.

For a complete discussion of the factoring algorithm see [4,5].




PREORDER: A B D E G C F

Figure 1

### 4.2. Edge Selection

In this section, results on optimal edge selection of Satyanarayana and Chang [10] are reviewed. Satyanarayana and Chang show that there exists an edge selection that yields the optimal binary structure, that is, a binary tree with the minimal number of leaves. They call such an edge selection the optimal edge selection. They also show that the number of leaves

of the optimal binary tree is equal to the domination, $D_K(G)$, of G. A *K-tree* is a tree of G covering all K-vertices and having its pendant vertices in $K$. An *irrelevant edge* is one that lies on no K-tree. An edge selection is optimal if and only if every reduced graph generated has no irrelevant edge. Hence, a fast edge selection strategy that avoids creating subgraphs with irrelevant edges is desired.

A graph $G$, with respect to some set $K$, is termed a *K-graph* if every edge of $G$ is relevant, i.e., if every edge of $G$ is in some K-tree of $G$. Satyanarayana and Chang prove that for a K-terminal irreducible graph G with domination $D_K(G)>1$, there exists an edge such that $G_e$ and $G_{-e}$ are both K-graphs. They further show, that an edge satisfying the property mentioned above, can be found in $O(|E| + |V|)$ operations using techniques based on depth-first-search [11,12].

In the version of the factoring algorithm implemented in *PolyChain* the optimal edge selection strategy is used. At each iteration an edge whose removal does not disconnect the graph is chosen. Then, each of the subgraphs generated by pivoting on the selected edge is checked for irrelevant edges. Notice that a graph is a K-graph if and only if each one of its pendant nonseparable components has at least one distinguished node. Therefore, the algorithm checks if the subgraphs are nonseparable. In the case a graph is separable, the algorithm checks if all of its pendant nonseparable components have at least one distinguished node in it. If a pendant nonseparable component not having any distinguished node exists in either one of the subgraphs generated by the edge selected, the current edge is discarded and another edge is selected to replace it. Then, the checking procedure starts all over again considering, now, the new edge selected. Since a graph having only one distinguished node is not a K-graph the algorithm avoids creating such graphs. This edge selection procedure is $O(|E|^2)$.

## 5. System Manual

In this section the code is briefly described.

### 5.1. Programming

All subroutines in this version on *PolyChain* are written in Fortran 77. Only I/O related code is system dependent. The input network for the factoring algorithm is the output of the original version of *PolyChain* when its input network is not totally reduced.

### 5.2. Data Structures

*PolyChain* uses an efficient network representation using linked list data structures, [6,7,8]. Each vertex has a list of adjacent vertices, which not only indicates which vertices are adjacent to it, but also provides information whether the vertex belongs to set K. For every element of the list, there is a pointer giving the address of the information about the edge. Figure 3 illustrates this multilist structure for the network given in figure 2.

The routines incorporated to *PolyChain* use a few other data structures in addition to the ones used by *PolyChain*. For nonseparability testing a stack data structure is used. For the factoring algorithm another stack is used to provide information about the computational binary tree.

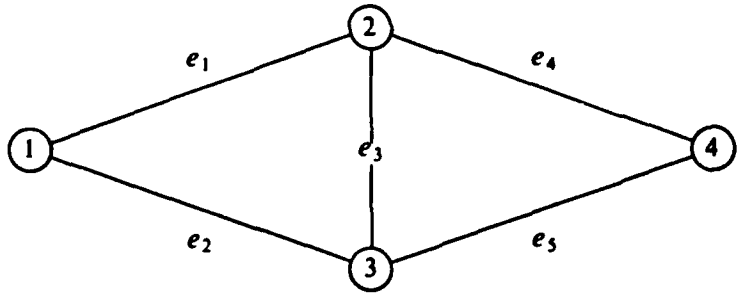



Figure 2

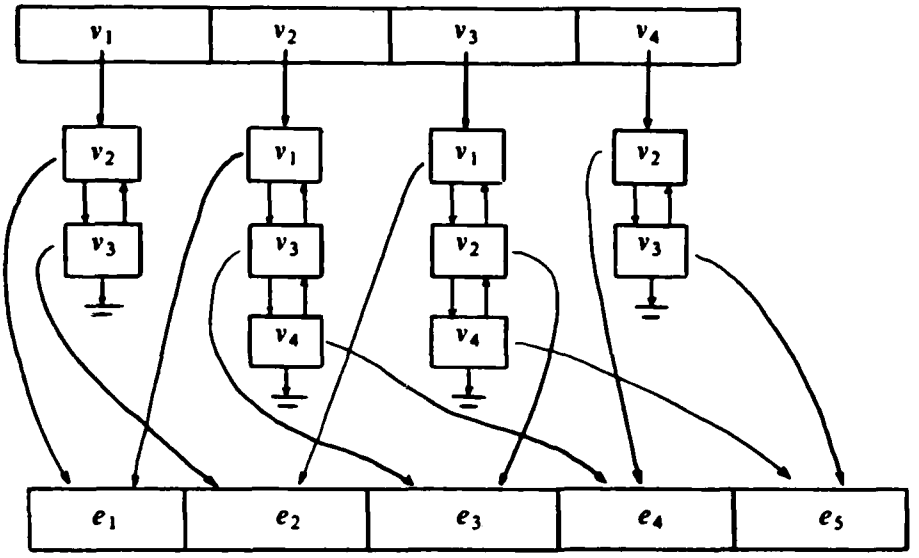



Figure 3

### 5.3. Data Structure Implementation

Next, we describe the FORTRAN arrays used to implement the data structure of the two routines.

#### 5.3.1. Nonseparability Test

As already mentioned a stack is used for producing the vertices of the component. The vertices are stored in the stack in the order that they are discovered. When a separating vertex $u$ is discovered, we read off all the vertices from the top of the stack down to a node specified by the algorithm. All these edges plus the separating vertex $u$ constitute the component.

#### 5.3.2. Factoring Algorithm

Five arrays are used to implement the stack that stores information about the computational binary tree. EDGEV1(*) and EDGEV2(*) contain the stack of vertices corresponding to the selected edge. DIRECT(*) contains information about the branching. If DIRECT(*) is 1, the selected edge is working. If it is -1, the selected edge is not working. EPROB(*) contains the reliability of the selected edge, and RELB(*) contains the value of M after all possible simple

reductions are performed. If no degree-2 reduction is performed, the value of M is 1. If a degree-2 reduction is performed, the value of M will be updated. TOP points to the top of stacks EDGEV1(*), EDGEV2(*), DIRECT(*), EPROB(*), and RELB(*).

The implementation of the factoring algorithm was carried out in a way to minimize core usage. As already mentioned, a preorder binary tree traversal algorithm was implemented. Hence, after an edge is selected we always consider first the case in which the selected edge is working. When the subgraph can be reduced by simple reductions, the algorithm finds its reliability and goes back to its parent node to continue branching. After leaving a node, which is actually a subgraph, that was already branched in both directions, the algorithm never comes back to it. Hence, this subgraph and all subgraphs beneath it do not have to be saved and can therefore be deleted. Since the factoring algorithm was not implemented in recursive form, we have to keep the information necessary for the recovery of the subgraph of the computational binary tree.

To better understand the method, suppose the computational binary tree is of the form given in figure 4.

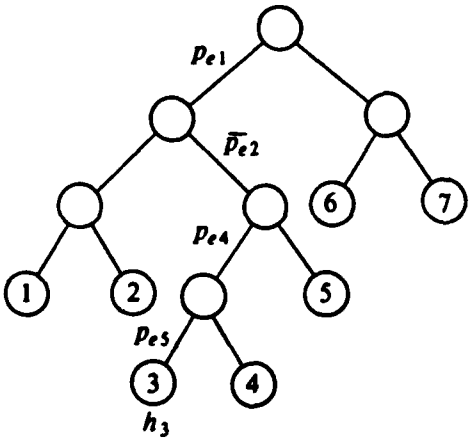
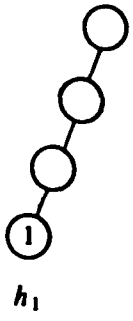


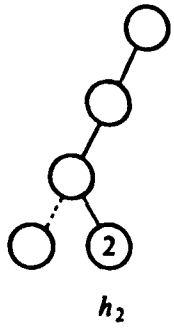

Figure 4

The numbered nodes correspond to series-parallel reducible subgraphs. The implemented factoring algorithm finds the reliability of each branch and then the reliability of the original graph as shown in figures 4a, 4b, 4c, 4d, 4e, 4f, and 4g. For example, if $h_i$ is the reliability of the branch leading to leaf $i$, the overall reliability of the original graph $G_K$ is $R_K(G) = M \prod_i h_i$, where $M = \prod_j \Omega_j$ obtained from the *polygon-to-chain* reductions. The reliability of a leaf, $h_i$, is the product of the reliabilities of the edges selected leading to that leaf. For example, $h_3 = p_{e1}\overline{p}_{e2}p_{e4}p_{e5}$.



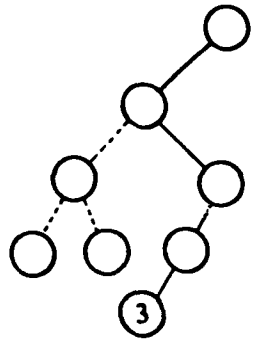

$h_1$

4.a



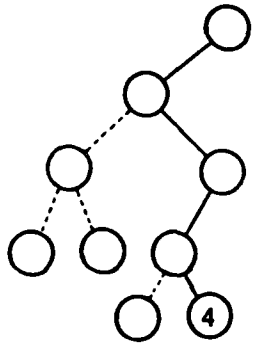

$h_2$

4.b



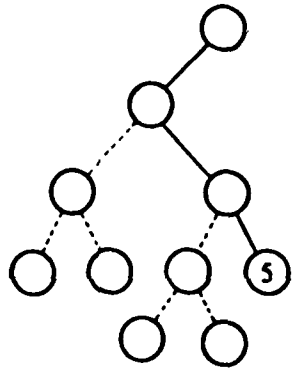

$h_3$

4.c



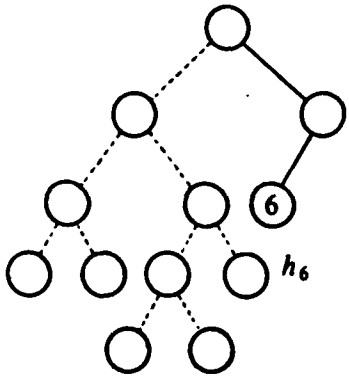

$h_4$

4.d



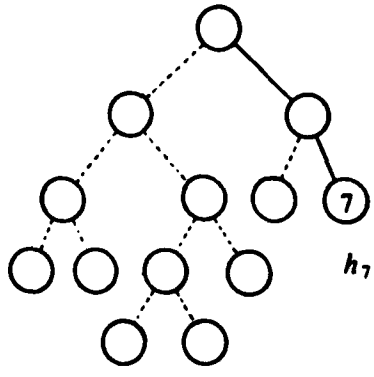

$h_5$

4.e




$h_6$

4.f




$h_7$

4.g

### 5.4. Data Dictionary

In this section, the variables used in the new subroutines are listed, with a brief description of each one. The variables marked with an asterisk are new variables, not present in the earlier version of *PolyChain*. A data dictionary containing other variables used in other *PolyChain* subroutines can be found in [1].

| | |
|---|---|
| ADJVRT(*) | vertex adjacent to vertex whose list it is on |
| *APTR | auxiliary pointer |
| *ATOPFB | auxiliary variable |
| *AUX | auxiliary variable |
| *AUX2 | auxiliary variable |
| *AUX3 | auxiliary variable |
| *AUXL | auxiliary variable |
| AVSADJ | pointer to beginning of list of available space |
| *BRIDGE(*) | array of edges whose removal disconnect the graph |
| CARDE | cardinality of set E |
| CARDV | cardinality of set V |
| *COUNTE | counter of the number of iterations |
| DATE | date |
| DEG(*) | degree of vertex |
| *DIR | auxiliary variable |
| *DIRECT(*) | direction of the branch taken |
| *DISTGN | current number of distinguished nodes |
| *DOMINT | domination of the graph |
| *EDGE | edge formed by the given pair of vertices |
| *EDGMRK(*) | array containing information about marked edges |
| *EDGEPV | edge that is a candidate to be a forbidden edge |

| | |
|---|---|
| *EDGEV1(*) | one of the vertices of selected edge |
| *EDGEV2(*) | one of the vertices of selected edge |
| EDGPRB(*) | edge reliability |
| *EPROB(*) | array of probabilities of selected edges |
| *FATHER(*) | preceding node in the search |
| *FNONSP | indicator of whether or not the output is to be printed |
| *FIRST | starting node in the search |
| *FIRSTV | vertex |
| *FORBED(*) | edges that are forbidden to be selected to be pivot |
| *FOUND | indicator of whether a new edge exists to continue search |
| *FOUND2 | indicator of whether a candidate degree-2 edge pair was found |
| *FOUNDP | indicator of whether parallel edges were found |
| *FOUNDS | indicator of whether series edges were found |
| *HEAD | auxiliary pointer |
| HOUR | hour |
| IN | value of FORTRAN input file |
| IOUT | value of FORTRAN output file |
| *IPTR | pointer |
| *K(v) | number of vertex v |
| *KGRAPH | indicator of whether or not a graph is a K-graph |
| *L(v) | lowpoint of v |
| *LIMITI | lower limit |
| *LIMITS | upper limit |
| LINECT | line counter |
| LNKDWN(*) | pointer to next element on list |
| LNKEDG(*) | pointer to corresponding edge |
| LNKUP(*) | pointer to element above in list |

| | |
|---|---|
| M | product of all Ω, see [3] |
| MAXLST | maximum number of elements in adjacent vertices list |
| MCARDE | cardinality of set E at start of procedure |
| MCARDV | cardinality of set V at start of procedure |
| ND2R | counter of degree-2 reductions |
| *NUMCMP | number of nonseparable components |
| *NUMELM | number of elements |
| *NPIVOT | number of pivots performed |
| *NPR | counter of parallel reductions |
| NSR | counter of series reductions |
| *OUTPUT | network reliability after factoring algorithm |
| *POINT | pointer |
| *PTR | pointer |
| PTRADJ(*) | pointer to beginning of list of adjacent vertices |
| *PTRCMP(*) | pointer to beginning of list of vertices in a component |
| QA | failure probability of edge a |
| QB | failure probability of edge b |
| *REL | total reliability |
| *RELB(*) | value of M after each simple reduction |
| *SECODV | vertex |
| *SEPVTX(*) | array of separable vertices |
| *SREL | subgraph reliability |
| *STACK(*) | stack of vertices scanned |
| *TADJVT(*) | copy of current adjvrt(*) |
| *TAVSAD | copy of current avsadj |
| *TBRIDG(*) | copy of current bridge(*) |
| *TCARDE | copy of current carde |

| | |
|---|---|
| *TCARDK | copy of current cardk |
| *TCARDV | copy of current cardv |
| TCPU | total cpu time |
| *TDEG(*) | copy of current deg(*) |
| *TEDGPB(*) | copy of current edgprb(*) |
| TEST | key for debugging feature |
| *TLNKDW(*) | copy of current lnkdwn(*) |
| *TLNKED(*) | copy of current lnkedg(*) |
| *TLNKUP(*) | copy of current lnkup(*) |
| *TM | copy of current m |
| *TMPCPU | solution time |
| *TOP | top of chain stack |
| *TOPB | pointer to the top of the list of bridges |
| *TOPCMP | pointer to beginning of the component |
| *TOPFB | pointer to the top of the list of forbidden edges |
| *TOPS | top of vertex stack |
| *TOPVTX | pointer to the top of the list of separable vertices |
| *TTOPB | copy of current topb |
| *U | vertex |
| *V | vertex |
| *V1 | vertex |
| *V2 | vertex |
| *VCMP(*) | vertex |
| VCPU | virtual cpu time |
| *VERTEX | vertex |
| *VERTX1 | vertex |
| *VERTX2 | vertex |

| | |
|---|---|
| VRTX | vertex |
| *XADJVT(*) | adjvrt(*) at start of factoring routine |
| *XAVSAD | avsadj at start of factoring routine |
| *XCARDE | carde at start of factoring routine |
| *XCARDV | cardv at start of factoring routine |
| *XDEG(*) | deg(*) at start of factoring routine |
| *XEDGPB(*) | edgprb(*) at start of factoring routine |
| *XLNKDW(*) | lnkdwn(*) at start of factoring routine |
| *XLNKED(*) | lnkedg(*) at start of factoring routine |
| *XLNKUP(*) | lnkup(*) at start of factoring routine |
| *XPTADJ(*) | ptradj(*) at start of factoring routine |
| YEAR | year |

### 5.5. COMMON Blocks

All the COMMON blocks used in the new routines are listed below. The COMMON blocks introduced in this new version of *PolyChain* are marked with asterisk. For a list of all other COMMON blocks in the code see [1].

```
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/BLK05/ MAXEDG,MAXVRT,MAXLST,MXSTKT,MAXCHN
COMMON/BLK06/ CARDE,CARDV,CARDK
COMMON/BLK07/ M
COMMON/BLK08/ IN,IOUT
COMMON/BLK31/ MCARDE,MCARDV,MCARDK
```

```
COMMON/BLK32/ DATE,YEAR,HOUR
*COMMON/BLK40/ K(MAXEDG),L(MAXEDG),STACK(MAXEDG),EDGMRK(MAXEDG),TOPS
*COMMON/BLK41/ PTRCMP(MAXVRT),TOPCMP,NUMCMP,VCMP(MAXEDG)
*COMMON/BLK50/ EDGEV1(MAXEDG),EDGEV2(MAXEDG),DIRECT(MAXEDG)
*COMMON/BLK51/ XPTADJ(MAXVRT),XADJVT(2*MAXEDG),XEDGPB(MAXEDG)
*COMMON/BLK52/ XLNKDW(2*MAXEDG),XLNKUP(2*MAXEDG),XLNKED(2*MAXEDG)
*COMMON/BLK53/ FOUND2
*COMMON/BLK54/ XDEG(MAXVRT),XCARDE,XCARDV,XAVSAD
*COMMON/BLK55/ EPROB(MAXEDG),RELB(MAXEDG),TOP,DISTGN
*COMMON/BLK57/ TPTADJ(MAXVRT),TADJVT(2*MAXEDG),TEDGPB(MAXEDG),TM,TTOPB
*COMMON/BLK58/ TLNKDW(2*MAXEDG),TLNKUP(2*MAXEDG),TLNKED(2*MAXEDG)
*COMMON/BLK59/ TDEG(MAXVRT),TCARDE,TCARDV,TCARDK,TAVSAD,TBRIDG(MAXEDG)
*COMMON/BLK60/ SEPVTX(MCARDV),TOPVTX
*COMMON/BLK61/ FORBEG(MAXEDG),BRIDGE(MAXEDG),TOPB,TOPFB
*COMMON/BLK84/ VCPU,TCPU
*COMMON/BLK98/ DOMINT,NPIVOT
*COMMON/BLK99/ COUNTE
```

### 5.6. Description of Subroutines

Next, the subroutines are presented and briefly described. Subroutines DELETE(V,PTR), SERIER(V), and DEG2R(V) are from the original version of *PolyChain*.

#### 5.6.1. SUBROUTINE NONSEP(FNONSP)

| | |
|---|---|
| Description | This subroutine finds the nonseparable components when the graph is separable, and the edges which removal disconnects the graph. |
| Input | The multilist structure and the logical variable FNONSP. |
| Output | A list containing the pointer to the beginning of the list of vertices of each |

component, the list of vertices of each component, and the list of bridges.

### 5.6.2. SUBROUTINE XPUSH(VERTEX)

Description This subroutine puts element VERTEX on the top of stack.

Input VERTEX and the stack.

Output The updated stack.

### 5.6.3. SUBROUTINE OUTSEP

Description Prints the output listing when the network is separable.

Input The list of vertices of each nonseparable component.

Output The vertices of each nonseparable component.

### 5.6.4. SUBROUTINE FACTOR

Description This subroutine controls the basic steps of the factoring algorithm.

Input The multilist structure of the reduced network obtained after polygon-to-chain reductions were performed.

Output The K-terminal network reliability.

### 5.6.5. SUBROUTINE REDUCE

Description This subroutine performs series, parallel, and degree-2 reductions.

Input The multilist structure.

Output The updated multilist structure after all possible simple reductions were performed.

### 5.6.6. 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.

### 5.6.7. 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.

### 5.6.8. SUBROUTINE COPY

Description This subroutine copies the current graph for later use.

Input The multilist structure, and the list of edges that are bridges for the current graph.

Output The multilist structure, and the list of edges that are bridges for the current graph.

### 5.6.9. SUBROUTINE SELECT(V1,V2)

Description This subroutine selects an edge to pivot.

Input The multilist structure, and the edges that are forbidden to be chosen.

Output The nodes that form the edge selected.

### 5.6.10. SUBROUTINE CHKKGR(KGRAPH)

Description This subroutine checks if the graph is a K-graph.

Input The multilist structure, and the list of vertices of each nonseparable component.

Output A logical variable indicating whether the graph is a K-graph or not.

### 5.6.11. SUBROUTINE GRAPHR(TOP)

Description This subroutine reconstructs a subgraph of the computational binary tree that is pointed to by TOP.

Input Pointer TOP and the multilist structure of the original graph.

Output The multilist structure of the reconstructed subgraph.

### 5.6.12. SUBROUTINE REMOVE(V1,V2)

Description This subroutine removes the edge incident to vertices V1 and V2 from the subgraph.

Input Vertices V1 and V2. The multilist structure.

Output The updated multilist structure, with the desired edge removed.

### 5.6.13. SUBROUTINE COLAPS(V1,V2,)

Description This subroutine changes the subgraph by considering the probability of the edge incident to vertices V1 and V2 as being equal to 1.

Input Vertices V1 and V2. The multilist structure.

Output The updated multilist

### 5.6.14. 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 commands.

Input The multilist structure. Vertex V. Pointer PTR.

Output The updated multilist structure without the specified element.

### 5.6.15. SUBROUTINE FNDREL(REL)

Description This subroutine computes the reliability of a reduced subgraph.

Input The stack defined by EDGEV1(*), EDGEV2, and DIRECT(*), and the current reliability REL.

Output Updated reliability.

### 5.6.16. SUBROUTINE FNDEDG(V1,V2,EDGE)

Description This subroutine finds the edge incident to vertices V1 and V2.

Input Vertices V1 and V2. The multilist structure.

Output The edge specified.

### 5.6.17. SUBROUTINE OUTFAC(OUTPUT)

Description This routine prints out the solution after the factoring algorithm was performed.

Input The current value of the reliability and the number of edges selected for pivoting.

Output The network's reliability and the number of edges selected.

## 6. User Manual

The user manual of this new version of *PolyChain* is similar to the manual of the original version [1]. In this section we first present a guide for using *PolyChain* showing the differences when using the VAX/UNIX system and the IBM/CMS system. The input file and output are then described, and a test problem is presented to illustrate outputs for both separable and nonseparable cases.

### 6.1. Executing Polychain

*Polychain* can be used in either the VAX/UNIX system or IBM/CMS system. As already mentioned, only I/O related code is system dependent. Therefore, to run the code, first the routine that gets the time, date, and day of the week from the system must be specified. Then, the dimension parameters and the COMMON blocks must be adjusted. Finally, an input data file must be prepared. These three topics are presented below.

#### 6.1.1. System Routines

The first step in running *Polychain* is adjusting the code to run in the desired system, either UNIX or CMS. To do this, the suitable system routine that gets the time, date, and day of the week must be specified. The code considers both possibilities, so that is just a question of removing or adding comments to the lines of the code where the system routines appear, depending on which one you need. The system routines are described below.

For IBM/CMS use:

MAIN ROUTINE:

```
CALL DATETM(DATTIM,23,VCPU,CTIME,TCPU)
DATE = DATTIM(1:16)
HOUR = DATTIM(19:23)
BEGINT = VCPU
```

SUBROUTINE OUTFAC, OUTGRF AND OUTREL:

```
CALL DATETM(DATTIM,23,VCPU,CTIME,TCPU)
TMPCPU = VCPU - TMPCPU
WRITE(IOUT,300) TMPCPU
```

For VAX/UNIX use:

MAIN ROUTINE:

```
CALL FDATE(ERA)
DATE = ERA(1:10)
HOUR = ERA(12:20)
YEAR = ERA(21:30)
CALL DTIME(TIME)
```

SUBROUTINE OUTFAC, OUTGRF AND OUTREL:

```
CALL DTIME(TIME)
WRITE(IOUT,300) TIME(1)
```

### 6.1.2. Dimension Parameters

The second step in running Polychain is adjusting the dimension parameters MAXVRT and MAXEDG in SUBROUTINE INILST. MAXVRT is the maximum number of vertices and MAXEDG is the maximum number of edges of the graph. The adjustment of these variables is needed only if the network's dimensions exceed what has been already specified.

After adjusting the dimension parameters, all COMMON blocks containing arrays must be changed accordingly. Section 4.5 shows how the arrays must be changed.

### 6.1.3. Input Files

Inputing data in *PolyChain* is very simple since 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. One value must be entered in this line - ECHOIN, where,

$$ECHOIN = \begin{cases} 1 & \text{if } a \textit{ report of the input network is desired} \\ 0 & \textit{otherwise} \end{cases}$$

Next, the edges are specified, one in each line. To specify an edge, enter both vertices of the edge followed by the edge's reliability. The numbering of the vertices should be sequential

from 1 to the number of vertices of the network. If a vertex is a K-vertex, it should be preceded by a minus sign. An example illustrating an input file is given below.

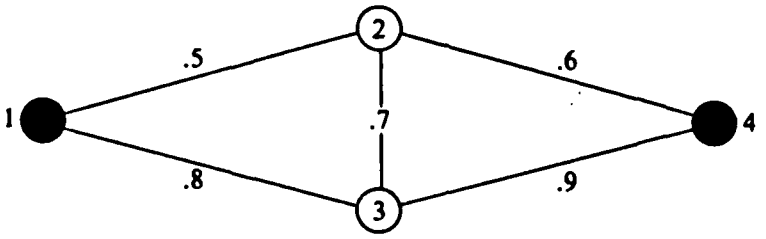



Figure 5

The input file for the network of figure 5, in the case that we want the input network report is given below.

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

### 6.1.4. Program Outputs

In this section a test problem is used to illustrate the program's output. Consider a series-parallel irreducible network, the ARPA computer network, in figure 6. The reliabilities (actually availabilities) shown in figure 6 are fictitious.

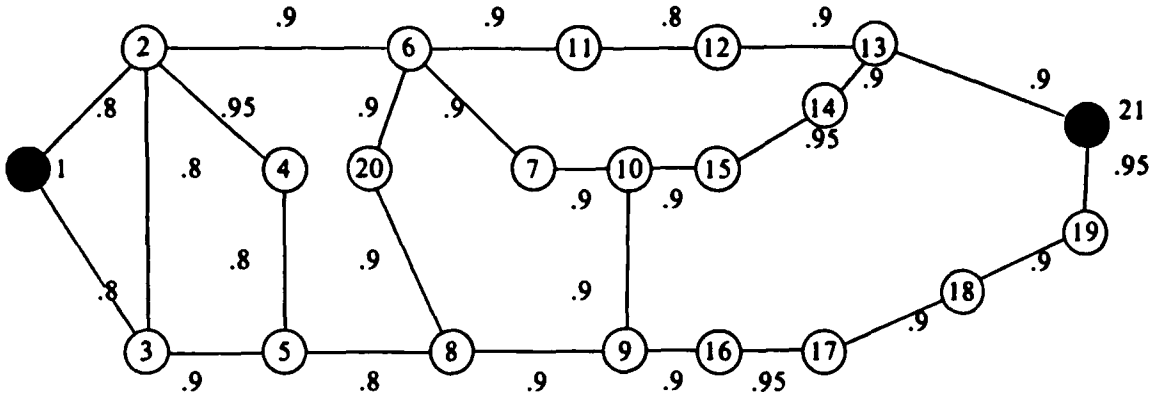



Figure III - ARPA Computer Network

The input file for this network is given next. Output option is set to "1".

```
1
-1    2    .8
-1    3    .8
2     3    .8
2     4    .95
2     6    .9
3     5    .9
4     5    .8
5     8    .8
8     20   .9
6     11   .9
6     20   .9
6     7    .9
7     10   .9
8     9    .9
9     10   .9
10    15   .9
11    12   .8
12    13   .9
14    13   .9
13    -21  .9
15    14   .95
9     16   .9
16    17   .95
17    18   .9
18    19   .9
19    -21  .95
```

*PolyChain* generates either a two part or a three part report depending on whether the network is series-parallel reducible or not, respectively. The first section of the report describes the input network, edge by edge. The type of each vertex is indicated, K for K-vertex and nK for non K-vertex. The first section also summarizes the input network data and core usage. Network density, presented in the first section, is defined to be the ratio of the number of edges of the input network to the number of edges of its corresponding complete graph. The second section of the report indicated whether the network is series-parallel reducible or irreducible. This section contains a summary of the reductions performed and the CPU time before the beginning of the factoring algorithm. In case the network is series-parallel irreducible, the updated value of $M = \prod_j \Omega_j$ is included in this section and the third part of the report is generated. The third section contains the K-terminal network reliability, the domination of the reduced network, the number of pivots performed, and the CPU time, excluding I/O. The report generated by *PolyChain* for the above file follows.

POLYCHAIN - VERSION 85.1 PAGE 1
POLYGON TO CHAIN REDUCTIONS
IN NETWORK RELIABILITY

DATE : FRI, NOV 29 1985
TIME : 15:08

INPUT NETWORK

| EDGE | VERTEX | TYPE | VERTEX | TYPE | RELIABILITY |
|---|---|---|---|---|---|
| 1 | 1 | K | 2 | NK | .80CCCOOOE+00 |
| 2 | 1 | K | 3 | NK | .80CCCOOOE+00 |
| 3 | 2 | NK | 3 | NK | .80CCCOOOE+00 |
| 4 | 2 | NK | 4 | NK | .95CCCOOOE+00 |
| 5 | 2 | NK | 6 | NK | .90CCCOOOE+00 |
| 6 | 3 | NK | 5 | NK | .90CCCOOOE+00 |
| 7 | 4 | NK | 5 | NK | .80CCCOOOE+00 |
| 8 | 5 | NK | 8 | NK | .80CCCOOOE+00 |
| 9 | 8 | NK | 20 | NK | .90CCCOOOE+00 |
| 10 | 6 | NK | 11 | NK | .90CCCOOOE+00 |
| 11 | 6 | NK | 20 | NK | .90CCCOOOE+00 |
| 12 | 6 | NK | 7 | NK | .90CCCOOOE+00 |
| 13 | 7 | NK | 10 | NK | .90CCCOOOE+00 |
| 14 | 8 | NK | 9 | NK | .90CCCOOOE+00 |
| 15 | 9 | NK | 10 | NK | .90CCCOOOE+00 |
| 16 | 10 | NK | 15 | NK | .90CCCOOOE+00 |
| 17 | 11 | NK | 12 | NK | .80CCCOOOE+00 |
| 18 | 12 | NK | 13 | NK | .90CCCOOOE+00 |
| 19 | 14 | NK | 13 | NK | .90CCCOOOE+00 |
| 20 | 13 | NK | 21 | K | .90CCCOOOE+00 |
| 21 | 15 | NK | 14 | NK | .95CCCOOOE+00 |

DATE : FRI, NOV 29 1985
TIME : 15:08

INPUT NETWORK

| EDGE | VERTEX | TYPE | VERTEX | TYPE | RELIABILITY |
|---|---|---|---|---|---|
| 22 | 9 | NK | 16 | NK | .90000000E+00 |
| 23 | 16 | NK | 17 | NK | .95000000E+00 |
| 24 | 17 | NK | 18 | NK | .90000000E+00 |
| 25 | 18 | NK | 19 | NK | .90000000E+00 |
| 26 | 19 | NK | 21 | K | .95000000E+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 |

DATE : FRI, NOV 29 1985
TIME : 15:08

NETWORK SERIES-PARALLEL IRREDUCIBLE
REDUCED NETWORK

| EDGE | VERTEX | TYPE | VERTEX | TYPE | RELIABILITY |
|---|---|---|---|---|---|
| 14 | 1 | K | 9 | NK | .86427719E+00 |
| 5 | 1 | K | 6 | NK | .98323840E+00 |
| 13 | 6 | NK | 10 | NK | .81CCCCCCE+0C |
| 18 | 6 | NK | 13 | NK | .64ECCCCCE+00 |
| 26 | 9 | NK | 21 | K | .65?92250E+CC |
| 15 | 9 | NK | 10 | NK | .90CCCCCCE+0C |
| 21 | 10 | NK | 13 | NK | .7655CCCCE+00 |
| 20 | 13 | NK | 21 | K | .9CCCCCCCE+00 |

POLYCHAIN - VERSION 85.1
POLYGON TO CHAIN REDUCTIONS
IN NETWORK RELIABILITY

DATE : FRI, NOV 29 1985
TIME : 15:08

UPDATED VALUE OF M = 0.93470698E+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 | 0.0 |

SOLUTION TIME = 0.00 SECS.

FACTORING ALGORITHM APPLIED

NETWORK RELIABILITY..... 0.86671016E+00

NUMBER OF BINARY TREE LEAVES: 7

DOMINATION: 4

SOLUTION TIME = 0.12 SECS.

## 7. Test Problems

Next, the results obtained by *PolyChain* applied to several networks are given. Some of the networks are obtained through a random network generator, while other tested networks are from [5]. Problems were run on the IBM 3081, at Berkeley. The code was compiled on the CMS FORTVS compiler using optimization level 3. CPU times were measured through the DATETM system routine. Table I contains a summary of the networks tested and table II a summary of test results. Figure 7 shows an example of a network where no polygon-to-chain reduction is possible for any set $K$ chosen.

Table I - Test Problems

| Problem | Vertices | Edges | K-Vertices | Type of Graph |
|---|---|---|---|---|
| 1 | 21 | 26 | 2 | ARPANET |
| 2 | 5 | 10 | 2 | Five Vertex Complete |
| 3 | 5 | 10 | 4 | Five Vertex Complete |
| 4 | 5 | 10 | 5 | Five Vertex Complete |
| 5 | 6 | 15 | 2 | Six Vertex Complete |
| 6 | 6 | 15 | 6 | Six Vertex Complete |
| 7 | 8 | 12 | 2 | Eight Vertex Cubic |
| 8 | 8 | 12 | 8 | Eight Vertex Cubic |
| 9 | 10 | 15 | 2 | Ten Vertex Cubic |
| 10 | 16 | 24 | 2 | Sixteen Vertex Cubic |
| 11 | 16 | 24 | 16 | Sixteen Vertex Cubic |
| 12 | 6 | 12 | 2 | Six Vertex Quartic |
| 13 | 6 | 12 | 4 | Six Vertex Quartic |
| 14 | 6 | 12 | 6 | Six Vertex Quartic |
| 15 | 20 | 59 | 4 | Random |
| 16 | 10 | 30 | 2 | Random |
| 17 | 15 | 39 | 2 | Random |

Table II - Test Results

| Problem | % Reduction | | | Domination | CPU Time |
|---|---|---|---|---|---|
| | Edges | Vertices | K-vertices | | |
| 1 | 69.2 | 71.4 | 0 | 4 | 0.12s |
| 2 | 0 | 0 | 0 | 6 | 0.18s |
| 3 | 0 | 0 | 0 | 6 | 0.18s |
| 4 | 0 | 0 | 0 | 6 | 0.09s |
| 5 | 0 | 0 | 0 | 24 | 0.72s |
| 6 | 0 | 0 | 0 | 24 | 0.36s |
| 7 | 0 | 0 | 0 | 16 | 0.48s |
| 8 | 0 | 0 | 0 | 11 | 0.15s |
| 9 | 0 | 0 | 0 | 40 | 1.40s |
| 10 | 0 | 0 | 0 | 448 | 17.86s |
| 11 | 0 | 0 | 0 | 247 | 2.38s |
| 12 | 0 | 0 | 0 | 11 | 0.36s |
| 13 | 0 | 0 | 0 | 20 | 0.17s |
| 14 | 0 | 0 | 0 | 11 | 0.19s |
| 15 | 50.9 | 30 | 25 | 5063 | 171.81s |
| 16 | 43.3 | 10 | 0 | 35 | 1.10s |
| 17 | 53.8 | 46.7 | 0 | 52 | 1.86s |

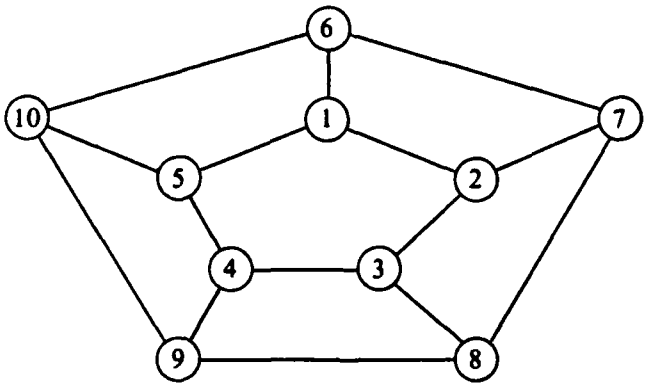



Figure 7 - Ten Vertex Cubic Graph

## 8. Conclusions and Recommendations

This report discussed the design and implementation of two features that enable *PolyChain* to treat a larger class of problems. The implementation of both features maintain the characteristics of the original version of *PolyChain* facilitating further extensions and enhancements.

Further testing is still needed to ensure the code's correctness.

To insure the evaluation of the K-terminal network reliability in a more efficient form, the program should apply polygon-to-chain-reductions in addition to simple reductions throughout the factoring algorithm.

In the case of separable networks a code using *PolyChain* as a subroutine can be used to compute the reliability of each nonseparable component and then compute the overall reliability of the network.

## 9. Acknowledgement

This research has been partially supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq, Brazil, and the Office of Naval Research, under contract N00014-85-K-0384.

## 10. References


[1] Resende, M., "A Computer Program for Reliability Evaluation of Large-Scale Undirected Networks via Polygon-to-Chain Reductions", ORC 83-10, Operations Research Center, University of California, Berkeley, 1983, submitted

[2] Satyanarayana, A., Wood, R.K., "A Linear Time Algorithm for K-Terminal Reliability in Series-Parallel Networks ", SIAM J. Computing, Nov. 1985.

[3] 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

[4] Agrawal, A., Barlow, R.E., "A Survey of Network Reliability and Domination Theory", Operations Research, vol 32,no 3,1984,pp 478-492

[5] Johnson, R., "Some Combinatorial Aspects of Network Reliability", Ph.D. Thesis, University of California, Berkeley, 1982

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

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

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

[9] Even, S., *Graph Algorithms*, Computer Science Press, 1979

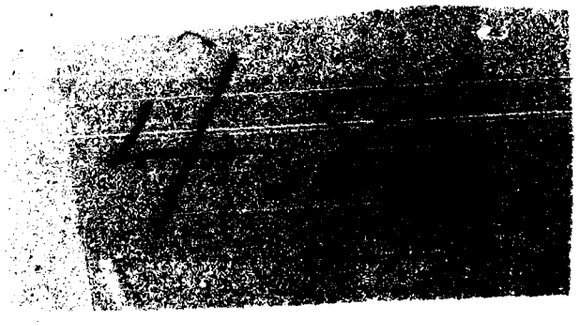
[10] Satyanarayana, A., Chang, M.K., "Network Reliability and the Factoring Theorem", Networks 13, 107-120, 1983.

[11] Hopcroft, J.E., Tarjan, R.E., "Dividing a Graph into Triconnected Components", SIAM J. Computing,2,135-158, 1973

[12] Tarjan, R.E., "Depth-First Search and Linear Graph Algorithms", SIAM J. Computing,1,146-160, 1972

END

FILMED



DTIC