Exclusive Tear Sets for Flowsheets

RODOLPHE L. MOTARD

Department of Chemical Engineering
Washington University
St. Louis, MO 63130

and

ARTHUR W. WESTERBERG

Department of Chemical Engineering
Design Research Center
Carnegie-Mellon University
Pittsburgh, PA 15213

A new implicit enumeration algorithm locates the minimum weighted tear sets among those which belong to the nonredundant families of Upadhye and Grens (1975). Theoretical developments support the algorithm and give a new insight relating nonredundancy to the tearing of unit loops in a flowsheet. If all loops can be torn exactly one time, the nonredundant family is unique and a member of it is trivial to find. If not possible, two or more nonredundant families exist and the above algorithm discerns among them, picking a tear set which minimizes the maximum number of times any unit loop is torn.

SCOPE

The literature abounds with algorithms to find the "tear" streams automatically for process flowsheeting calculations. A review article by Motard, Shacham and Rosen (1975) discusses most such algorithms. In a flowsheeting calculation, n equations, such as the heat and material balance relationship, equilibrium and rate relationships, are solved for a process flowsheet to yield values for an equal number of unknowns such as temperatures, pressures, flows and stream compositions. In a sequential modular type of flowsheeting system such as FLOWTRAN (Seader, Seider and Pauls, 1974; CONCEPT, 1973; or PACER, 1971, each unit is modeled by a subroutine which calculates output stream variables given values for all input stream variables and all equipment size parameters for the unit. For this type of approach the calculations for a flowsheet follow the material flows through the process from given process input stream values, around recycle loops if they exist and finally to output stream values.

A first step to finding an efficient computational sequence through the unit subroutines is to locate the "partitions" of a flowsheet, that is, finding the groups of units which are tied together through process recycles and thus which have to be solved together. Tear streams are those which are to be guessed and iterated in the course of solving a partition. One criterion for selecting tear streams is to select the fewest such streams. Of many published algorithms, the one of Barkley and Motard (1972) effectively solves this problem. Another criterion is to select the minimum weighted tear set where each stream is assigned a weight, and the best tear set is defined as the one which gives rise to the minimum sum of weights associated with the tear streams. Christensen and Rudd (1969) present an effective approach for this criterion.

The best criterion appears to be that of Upadhye and Grens (1975) wherein the tear set is required to belong to a "nonredundant" family of tear sets. All tear sets in this family are shown to have the same convergence behavior using successive substitution, and Upadhye and Grens give qualitative arguments, together with numerical evidence, that these tear sets are likely to be better than any others. Rosen and Pauls (1977) support this contention with further numerical evidence using the well-known Cavett problem (Cavett, 1963).

The purpose of this paper is to prove certain characteristics about the nonredundant family of tear sets of Upadhye and Grens and thus provide new insights into finding such sets. It becomes possible to distinguish among classes of nonredundant families which were observed but not explained by Upadhye and Grens. For most flowsheeting problems, these insights give a trivially easy algorithm to find a nonredundant tear set.

CONCLUSIONS AND SIGNIFICANCE

We have shown in this paper that the nonredundant family of tear sets is directly related to the unique tearing of unit loops within the flowsheet. We discovered that the inability to find a tear set which tears all unit loops exactly one time gives rise to more than one nonredundant family in the Upadhye and Grens sense. Such a discovery allowed us to distinguish among nonredundant families and argue qualitatively that some of these should have better convergence properties than others.

Finally, we gave and demonstrated an algorithm to find the minimum weighted tear set from among those in the nonredundant families deemed best by the above arguments. If only a member of a nonredundant family is desired and that family is unique, we described a trivially easy algorithm to find it, provided the unit loops are available.

The algorithm presented in this paper has been used as the flowsheet tearing algorithm for the ASPEN program being developed at MIT [DOE Contract No. E(49-18)-2295, Task No. 9] and has been tested on hundreds of practical examples.

0001-1541-81-4798-0725-$2.00.

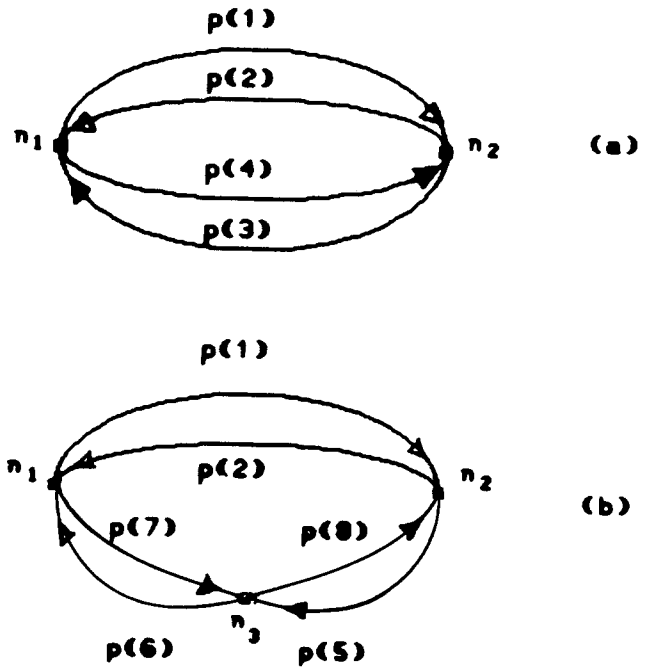

Figure 1. Minimum structure required in a digraph for a covered loop to exist.

If no edge in $v(i)$ remains unflagged, we shall define $v(i)$ as a covered node loop. Clearly then $v(i)$ can be a covered node loop if and only if a subset of k of the (tear) edges selected by Algorithm I have the following properties.

1) None of them is in a simple node loop with any of the rest of them.

2) Each tear edge is responsible for flagging some of the edges in $v(i)$ by being in simple node loops with these edges.

3) As a set they must cause all edges in $v(i)$ to be flagged.

We will now examine the minimum structure which must be present in a digraph for it to contain a node loop. See Figure 1a. Consider two nodes n_1 and n_2 connected by the paths $p(1)$ and $p(2)$. Path $p(1)$ goes from n_1 to n_2 and $p(2)$ from n_2 to n_1. Let $v(1) = p(1)\,p(2)$ be a covered node loop. We must be able to locate a second simple node loop, $v(2)$, which contains $p(1)$ but not $p(2)$; therefore we need a second path $p(3)$ from n_2 to n_1 with $v(2) = p(1)\,p(3)$ resulting. In addition we need a simple node loop, $v(3)$, containing $p(2)$ and neither $p(1)$ nor $p(3)$. We therefore require a second path $p(4)$ from n_1 to n_2 with $v(3) = p(2)\,p(4)$, but we find we have also formed the node loop $v(4) = p(3)\,p(4)$. Selecting an edge from $p(3)$ will cause us to flag all edges in $p(1)$ but will also cause us to flag all edges in $p(4)$. Clearly both $p(3)$ and $p(4)$ must be created if $v(1)$ is to be covered. The node loop $p(3)\,p(4)$ must result; it cannot be a simple node loop therefore (a simple node loop contains no other node loops). Paths $p(3)$ and $p(4)$ must pass through a common third node, n_3, which breaks $p(3)$ into two

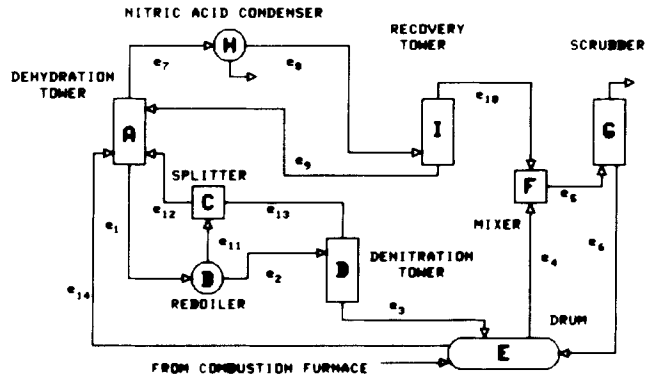

Figure 2. An example flowsheet (Newman and Klein, 1972).

TABLE 1. NODE LOOPS ($v_{(1)}$ TO $v_{(6)}$) DISPLAYED AGAINST EDGES IN FLOWSHEET IN FIGURE 2. BOTTOM PART OF TABLE SHOWS FLAGS SET USING ALGORITHM I WHEN EDGES e_1, e_4 AND THEN e_7 ARE SELECTED. CIRCLED INCIDENCES IN TOP HALF SHOW HOW EACH LOOP IS TORN BY EXACTLY ONE EDGE FROM THE EDGE TEAR SET.

Node Loops \ Edges	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}	e_{13}	e_{14}
$v(1)$				Ⓧ	x	x								
$v(2)$	Ⓧ	x	x											x
$v(3)$	Ⓧ										x	x		
$v(4)$	Ⓧ	x	x										x	x
$v(5)$							Ⓧ	x	x					
$v(6)$					x	x	Ⓧ	x		x				x
Resulting Flags														
Select e_1	*	*	*								*	*	*	*
Select e_4				*	*	*								
Select e_7							*	*	*	*				

paths, $p(5)$ and $p(6)$, and $p(4)$ into $p(7)$ and $p(8)$ as illustrated in Figure 1b. The node loop $p(3)\,p(4)$ then contains two node loops and is no longer simple.

The simple loops for Figure 1b are $[p(1)\,p(2)]$, $[p(6)\,p(7)]$, $[p(5)\,p(8)]$, $[p(1)\,p(5)\,p(6)]$ and $[p(2)\,p(7)\,p(8)]$. To cover loop $v(1) = p(1)\,p(2)$ we select one edge each from $p(5)$ and $p(7)$ [or from $p(6)$ and $p(8)$]. Selecting an edge from $p(5)$ flags all edges in $p(1)$, $p(5)$, $p(6)$, and $p(8)$. Then selecting an edge from $p(7)$ flags all edges in $p(2)$ and $p(7)$. Thus, $v(1) = p(1)\,p(2)$ is indeed covered.

Result 1: If a digraph contains no covered loops, an edge tear set can always be developed which tears each node loop exactly one time.

Proof: Apply Algorithm I to the digraph. Algorithm I can terminate in only one of two ways: (1) it terminates with all node loops torn exactly one time; or (2) it terminates with one or more covered node loops left untorn. By assumption the digraph contains no covered node loops. Thus Algorithm I terminates by option (1) and the edge tear set developed by the algorithm is a tear set of the type desired.

We shall call such a tear set when it exists an *exclusive edge tear set.*

We will use the process flowsheet in Figure 2 to illustrate how simply Algorithm I finds an exclusive edge tear set when no covered node loops exist in the flowsheet. Table 1 lists all the simple node loops $v(1)$ to $v(6)$ for the flowsheet in terms of the edges e_1 to e_{14}. For example node loop $v(1)$ involves edges e_4, e_5 and e_6. At the lower half of Table 1 are the flags which are set as we apply Algorithm I, selecting first edge e_1 (sets flags for edges e_1, e_2, e_3, e_{11}, e_{12}, e_{13} and e_{14}), then edge e_4 (sets flags for edges e_4, e_5 and e_6) and finally e_7 (which set flags for all remaining edges). We discuss later how to find all the simple nodes loops for a flowsheet.

We will now note some properties of the structure in Figure 1b which are required for a covered node loop to exist. At least three nodes must exist with each having at least two input edges and two output edges.

Result 2: If a digraph $G(N,E)$ contains fewer than three nodes, each with both multiple inputs and outputs, it cannot contain a covered node loop and an exclusive tear set can always be found.

Proof: The proof is a direct consequence of the above observation. Q.E.D.

We shall call the structure in Figure 1b a *cyclic cascade* and note that if and only if one exists in a digraph, then so does a covered node loop. Note the flowsheet in Figure 2 does not contain a cyclic cascade. Apparently such a structure is not commonly found in the digraph corresponding to a process flowsheet so an exclusive edge tear set exists for the digraphs corresponding to most flowsheets and is readily found by Algorithm I. Anticipating the connection between exclusive tear sets and unique nonredundant families, we note that Upadhye

and Grens discovered but one flowsheet with more than one nonredundant family out of several hundred tested, and this flowsheet was highly heat integrated. (Heat exchangers have multiple input and output edges.)

We now wish to prove the following theorem.

Theorem 1: If and only if a digraph $G(N,E)$ has an exclusive tear set, then the nonredundant family of edge tear sets as defined by Upadhye and Grens is unique, and each member of it is an exclusive tear set.

Proof: IF PART: We shall first need to define a nonredundant family of edge tear sets which we shall do using the Upadhye and Grens replacement rule. This rule states that an edge tear set is transformed to another in the same family by first identifying a node which has all of its input edges in the tear set. These edges are deleted and replaced by all the output edges of that node. The family is nonredundant if, after exhaustive application of the replacement rule, no edge already in the tear set is also introduced by the replacement rule. This double listing of an edge is redundant, and the algorithm to find a nonredundant family by this approach states that one should delete all but one listing of the repeated edge, moving to a new family and then continue. The above steps are repeated until a family is found which is shown to be nonredundant.

Let us assume an exclusive tear set exists. Then each simple node loop is torn exactly one time (see Result 1 and definition of an exclusive tear set). The replacement rule replaces all of a node's input edges by its output edges. All simple node loops passing through that node are torn once by these input edges before the replacement by assumption. Clearly they are *all* torn exactly once after the replacement by the output edges. Thus if the replacement rule starts with an exclusive edge tear set, it can only generate exclusive edge tear sets.

THEORY

The structure of a flowsheet can be captured in an obvious manner using graph theoretic concepts. We shall assume we are dealing with an irreducible subset of units within the flowsheet, wherein all the units in the subset must be solved together because of recycles. We shall make the following standard definitions to aid us.

A *digraph* $G(N,E)$ is a directed graph comprising a set of nodes and directed edges connecting those nodes, where

$$N = \{n_i | n_i \text{ is a node in } G\}$$
$$E = \{e_j | e_j \text{ is a directed edge in } G \text{ running from a source node } n_s \text{ to a destination node } n_d,\ n_s,\ n_d \text{ are members of } N\}$$

Clearly the nodes correspond to the process units in a process flowsheet and the directed edges to the connecting process streams.

A *path* $p(l)$ is a string of nodes and edges in G of the form:

$$p(l) = n_1\ e_1\ n_2\ e_2 \ldots n_m\ e_m\ n_{m+1}$$

where n_k is the source node and n_{k+1} is the destination node for edge e_k.

A *node loop*, $v(l)$, is a path where $n_1 = n_{m+1}$.

A *simple node loop* is a node loop which does not contain two or more node loops within it.

An *edge tear set* $ET(l)$ is a set of edges with the properties.

1) If the edges in $ET(l)$ are deleted, G will contain no node loops.

2) If any single edge in $ET(l)$ is not deleted while the remaining ones are, G will contain node loops.

We shall now define a *covered node loop* $v(i)$, where $v(i)$ is a simple node loop in G. To do this, we present the following algorithm.

Algorithm I

I. Select any (tear) edge e_μ which is *not* a member of $v(i)$.

II. Flag edge e_μ and all edges which are in any simple node loop with it. Some of these flagged edges may be in $v(i)$.

III. Repeat, selecting another (tear) edge e_μ which is *not* a member of $v(i)$ and is not yet flagged. Continue until the edges in all other node loops with any edges in common with $v(i)$ are flagged.

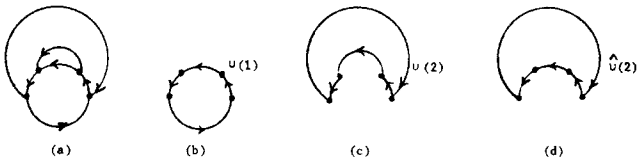

Figure 3. Finding a loop satisfying property 3.

We shall next prove that, if an exclusive tear set exists, repeated application of the replacement rule will transform it into *all* others. Assume that exclusive edge tear set $ET(1)$ exists. Generate any other by use of Algorithm I and call it $ET(2)$. We shall show that one can transform $ET(1)$ into $ET(2)$ by systematic application of the replacement rule, and, since $ET(1)$ and $ET(2)$ are arbitrary exclusive tear sets, we prove the "If Part" of our theorem. Order the simple node loops, and then, for each such loop, identify the edge which tears it in $ET(1)$ and the edge which tears it in $ET(2)$. We apply the following algorithm.

I. Select a simple node loop and call it $v(1)$.

II. Move the tear for $v(1)$ forward around $v(1)$ via the replacement rule until either (A) the tear for the loop reaches the desired location for it in $ET(2)$ or (B) a node $n(1)$ is encountered with multiple inputs.

A. If (A) is true, select a node loop whose tear is not yet in the position desired in $ET(2)$ and repeat from II. Call this node loop $v'(1)$.

B. If (B) is true, continue with step III.

III. Discover a loop $v(2)$ with the properties (1) the loop passes through $n(1)$, (2) the loop is not torn by the tear for $v(1)$ and (3), if $v(1)$ and $v(2)$ have common portions, these common portions are connected. [If a node loop exists which satisfies properties (1) and (2) but not (3), then another node loop exists which satisfies property (3), and it is to be the one selected. Figure 3 presents an example. Figure 3a shows our diagraph. Figure 3b identifies node loop $v(1)$ and Figure 3c, node loop

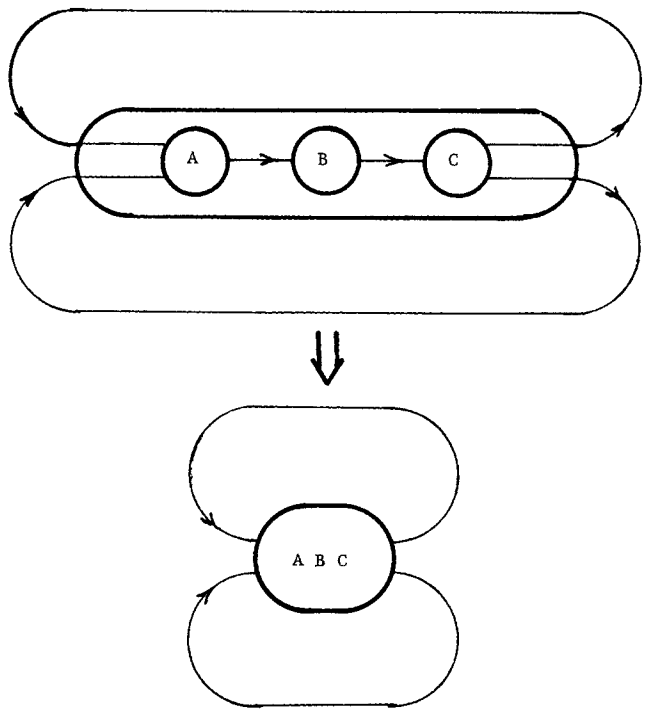

Figure 4. Merging common portions of two loops into a supernode.

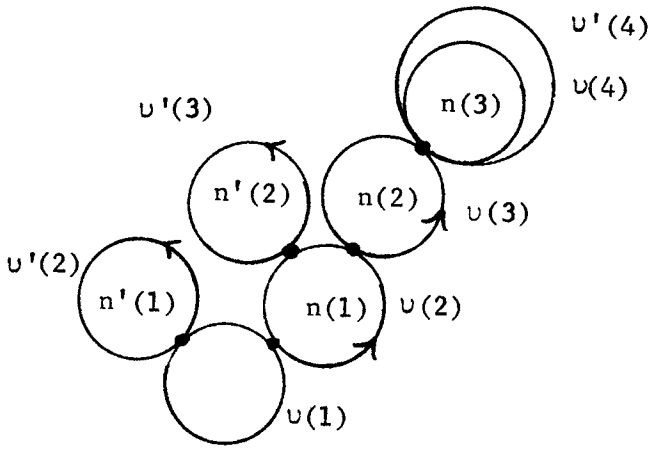

Figure 5. Tree structure to the loops discovered when moving tears for $\upsilon(i)$.

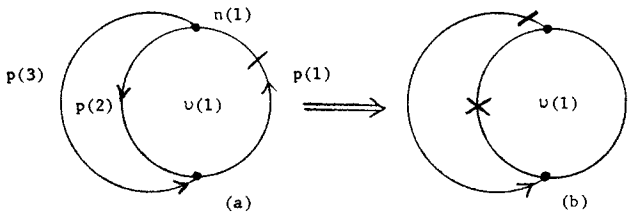

Figure 6. Moving a tear in $\upsilon(1)$ through a diverging node.

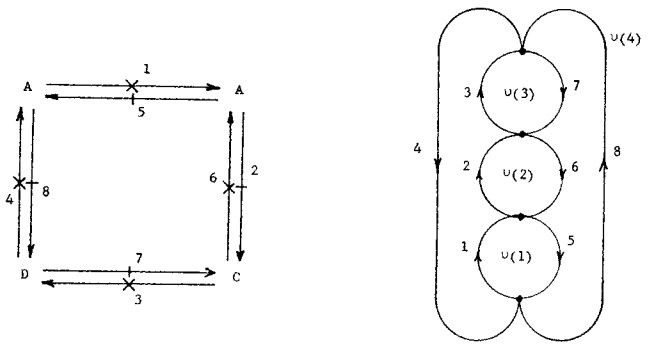

Figure 7. A digraph leading to a nontree loop structure.

$\upsilon(2)$. Note that $\upsilon(2)$ has two disconnected portions in common with $\upsilon(1)$. For this example we replace the parallel path in $\upsilon(2)$ which connects the two common portions by the part of $\upsilon(1)$ existing between these portions and find a loop $\hat{\upsilon}(2)$ which satisfies property 3.]

IV. Merge portions of $\upsilon(1)$ and $\upsilon(2)$ which are in common, as illustrated in Figure 4, into one supernode. Clearly the tear for $\upsilon(2)$ cannot be in the common portion for otherwise $\upsilon(1)$ would be doubly torn.

V. Starting now with $\upsilon(2)$, move the tear forward via the replacement rule until either (*A*) node $n(1)$ (which may now be in a supernode) is encountered or (*B*) a node $n(2)$ is encountered which has multiple inputs.

A. If (*A*) is true, then

1. Repeat from III if $n(1)$ has an as yet untorn input edge. Call the new loop found $\upsilon'(2)$.

2. Otherwise all edges to $n(1)$ are now in the tear set so we can move the tear set through $n(1)$ via the replacement rule. If $n(1)$ is in a supernode, unmerge the supernode and move the tear set through $n(1)$ via the replacement rule. Continue with step II.

B. If (*B*) is true continue with Step VI.

VI. Find a loop $\upsilon(3)$ with the properties (1) the loop passes through $n(2)$, (2) the loop is not torn by the tears for $\upsilon(1)$ nor $\upsilon(2)$ and (3), if $\upsilon(1)$, $\upsilon(2)$ and $\upsilon(3)$ have common portions, these portions are connected.

VII. Merge portions of $\upsilon(3)$, $\upsilon(2)$ and/or $\upsilon(1)$ which are in common into one supernode. [If the supernode includes parts of $\upsilon(1)$, then in fact $\upsilon(3)$ is not separated from $\upsilon(1)$ by $\upsilon(2)$. It should be relabeled as loop $\upsilon'(2)$, a "second (not third) level" node loop relative to $\upsilon(1)$.]

VIII. Start now with $\upsilon(3)$ and move its tear forward via the replacement rule until either (*a*) node $n(2)$ is encountered or (*b*) a node $n(3)$ is encountered which has multiple inputs, etc.

It should be clear with the above how to continue. The loops $\upsilon(1)$, $\upsilon(2)$, ... can be discovered only to a finite depth because each must not be torn by the tears of the earlier ones and only a finite number of simple loops occur in a finite digraph.

The loops $\upsilon(1)$, $\upsilon(2)$, ... from a "tree" of loops with $\upsilon(1)$ being the root. Figure 5 illustrates. The structure must be a tree for, if it is not, then it will contain a covered loop and an exclusive tear set does not exist. Loops $\upsilon'(k)$ and $\upsilon(k)$ may contain common edges and nodes, but, since they are not dealt with at the same time, no merging need occur among them unless they both join loop $\upsilon(k-1)$ at the same node (or supernode).

We now argue that, since the loops form a tree structure, we can move the tear for loop $\upsilon(1)$ to its position in $ET(2)$, and then we can return the tears for all loops above $\upsilon(1)$ in the tree to their original positions unless the tear for $\upsilon(1)$ in $ET(2)$ is in the portion of $\upsilon(1)$ and $\upsilon(2)$ which is in common. The tear for $\upsilon(2)$ in $ET(2)$ must be that for $\upsilon(1)$ in this case, and it need not be moved.

Assume the tear is not in the common portion. Clearly the tear for $\upsilon(1)$ can be moved forward via the replacement rule through $n(1)$ and then $n'(1)$ if necessary to get it to the position desired in $ET(2)$. After moving through $n(1)$ the tear for $\upsilon(2)$ just follows $n(1)$. It can be moved forward anywhere around $\upsilon(2)$ without encountering $n(1)$ again, and thus the tear for $\upsilon(2)$ can be put back to its original position without moving the tear for $\upsilon(1)$. Proceeding up the tree to level 3 the tear for $\upsilon(3)$ cannot be in the common section for loops $\upsilon(1)$, $\upsilon(2)$ and $\upsilon(3)$ for if it were then $\upsilon(3)$ would have become a loop at level 2 in our tree. Thus, this tear can always be returned to its original position, and it can be done without disturbing $\upsilon(1)$ or $\upsilon(2)$, etc.

While we can see that all tears for the loops in the tree structure can be returned to their original places, except perhaps for $\upsilon(2)$ where it is not necessary if it cannot be done, tears for other loops not in the tree may have been moved because of the moving of the tear for $\upsilon(1)$.

Such tears must come because the tear for $\upsilon(1)$ passes through a "diverging" node $n(1)$ which has an edge leaving it which is not a part of any of the loops appearing in the tree of loops. Since all paths leaving $\upsilon(1)$ must return to $\upsilon(1)$ eventually [because $G(N,E)$ is irreducible], such a path must belong to one (or more) loops which are torn by the tear for $\upsilon(1)$ in $ET(1)$ but not in $ET(2)$. Otherwise the edge would appear in the tree. In other words the edge starts one of more paths which leave $\upsilon(1)$ at node $n(1)$ to return at a one of more nodes, all of which follow the desired tear for $\upsilon(1)$ in $ET(2)$. Figure 6 illustrates. The original tear for $\upsilon(1)$ is shown in Figure 6a, the final desired tear in Figure 6b.

We note that in passing the tear through node $n(1)$, a tear is created along the parallel path at its beginning. Clearly the path must have a tear in $ET(2)$ since the path $p(1)$ is covered by the tear for $\upsilon(1)$ and the loop $p(1)p(3)$ must therefore be torn along $p(3)$. Having the tear at the beginning of $p(3)$ guarantees us that the tear in $p(3)$ can be moved anywhere along $p(3)$ and thus to its desired position in $ET(2)$ without affecting the tear for loop $\upsilon(1)$.

The approach is therefore to move the tear for each loop in turn using the above algorithm. Tears are created for parallel paths at their beginning so they may then be moved forward to exactly where needed. Tears in loops created above the loop of interest in the tree structure can always be returned to their original position so previously moved tears can be moved back to their target position if they appear later in the upper levels of a tree for a different loop.

Thus we can take each loop in turn and place the tear for it from $ET(1)$ to its position in $ET(2)$ without moving previously moved tears, and this position may be anywhere in the loop. The digraph has a finite number of simple loops so the process is

finite. Thus the replacement rule will generate all the exclusive tear sets starting from any given one.

ONLY IF PART: The proof depended on the nonexistence of a covered loop for otherwise the loop tree in Figure 5 would cease to be a tree. A loop at a higher level would find itself connected to a loop at a lower level as illustrated in Figure 7.

The tear set indicated by the single strokes cannot be transformed into the one indicated by the x's by systematically using the replacement rule, *QED*.

An example will illustrate the ideas involved in Theorem 1 and its proof. Consider the graph in Figure 8a. Select $v(1)$ (Figure 8b) as the paths {1,2,7} and move the tear from path 2 to just before node C, a node with two inputs. We discover $v(2)$ an untorn loop passing through node C. It has common edges 1 and 7 with $v(1)$ so Figure 8d is formed by merging these edges, forming a single supernode (comprising nodes C,A,B) which joins the two loops $v(1)$ and $v(2)$.

The tear for loop $v(2)$ is moved to just before supernode C,A,B. Note it creates a tear at the start of path 5 because diverging node D has two output edges. Figure 8e shows the result of expanding the supernode and then applying the replacement rule across node C, creating a tear on paths 4 and 7. We find node A on $v(1)$ has two inputs so we must locate a loop $v'(2)$ (Figure 8f) which is not torn and which passes through C. Figure 8g shows the tree of loops with common edges 1 and 2 merged. Node E has two inputs, so we find loop $v(3)$ but it has edges 8 and 1 in common with both $v(2)$ and $v(1)$. Merging these edges yields the tree in Figure 8i. We see that our supernode is nodes $EABC$ combined and that $v(3)$ really is a second loop tied to loop $v(1)$ at the supernode; $v(3)$ is relabeled $v''(2)$ since it is at the second level only of the tree.

We can now move the tears on 9 and 4 through node E to path 8. At this point both paths into node A are torn, and we move our tears through A onto path 1 (Figure 8l), our desired goal for the tear for loop $v(1)$. Since the tear on path 1 is in the common part of $v(1)$ with $v(2)$, $v'(2)$ and $v''(2)$, these loops are torn, and we cannot put the tears along these loops back to their original positions—nor should we.

We finally find loop $v'(1)$ comprising paths 3 and 5 but the tear is already on path 5 from when we moved the tear for loop $v(2)$ through node D. Thus, this tear is already in place. Figure 8n shows the final tear set, $ET(2)$, imposed on our network.

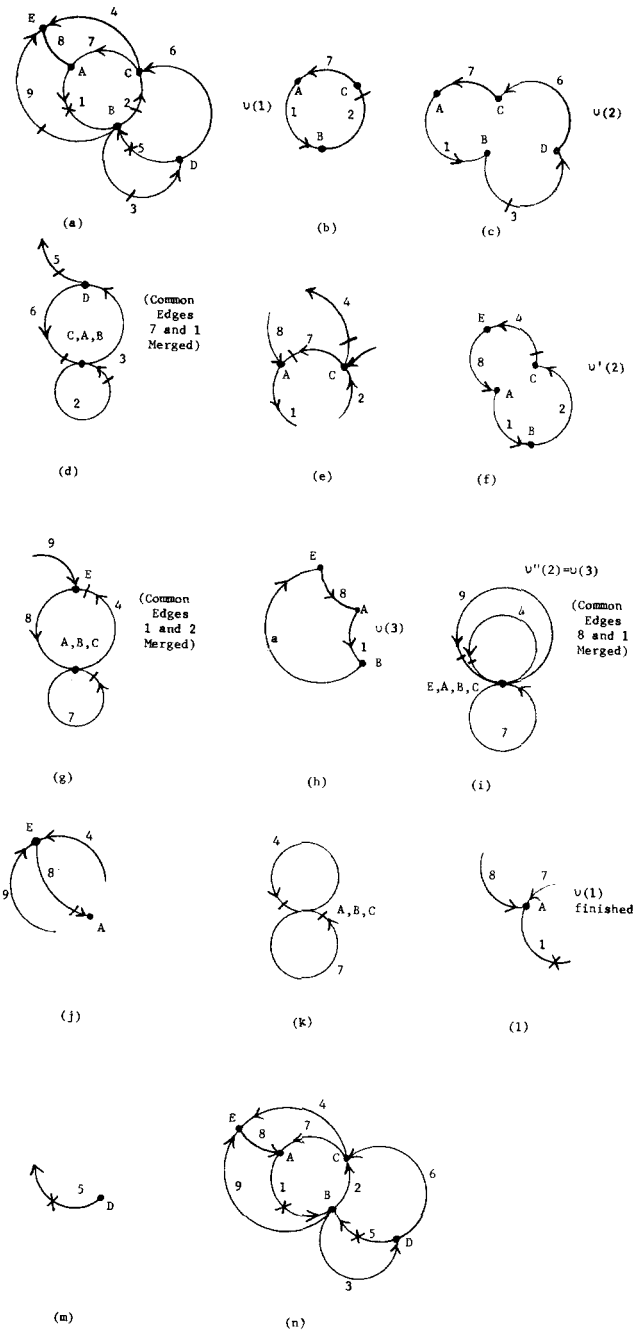

Figure 8. Moving tears from $ET(1) = 2,3,9$ to $ET(2) = 1,5$.

NEW TEAR ALGORITHM

We extend the argument of Upadhye and Grens by favoring those tear schemes which tear unit loops the fewest number of times. We would prefer to select tear schemes which minimize the maximum number of times any loop is torn. If an exclusive tear set exists, we want our best tear set to be an exclusive tear set. If multiple tears must occur for some of the loops, then we prefer the edge tear sets which only doubly tear node loops to those which might triple tear one or more node loops, etc.

We examine the digraph in Figure 7 to illustrate what we mean. This digraph contains a looping cascade and thus has no exclusive tear set. We display the simple node loops for this digraph by using a loop/edge incidence matrix. We list each loop along the left border of the matrix and each edge across the top. If an edge appears in the loop, we put a nonblank character (e.g. 'x') in the row for the loop. For the digraph in Figure 7, the loop incidence matrix is:

Loop	Edge	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
$v(1)$		x				x			
$v(2)$			x				x		
$v(3)$				x				x	
$v(4)$					x				x
$v(5)$		x	x	x	x				
$v(6)$						x	x	x	x

We can first apply Algorithm I and discover a covered loop. We first remove edge 1 which covers (flags) edges e_1, e_2, e_3, e_4 and e_5 and tears loops $v(1)$ and $v(5)$. We then remove edge e_6 which covers edges e_7 and e_8 and tears loops $v(2)$ and $v(6)$. Clearly loops $v(3)$ and $v(4)$ are covered but not torn. The Upadhye and Grens replacement rule would find three different nonredundant families:

1) based on one tear of loop $v(5)$ and three of loop $v(6)$.
2) based on two tears each of loops $v(5)$ and loop $v(6)$.
3) based on three tears of loop $v(5)$ and one of loop $v(6)$.

We see that families 1 and 3 triple tear a loop whereas family 2 double tears two loops.

The algorithm we wish to propose will select the second family as best because no node loop is torn more than twice for it.

We now devise an algorithm based on implicit enumeration (branch and bound) to locate effectively an edge tear set for an irreducible graph. We shall allow an edge e_j to have a weight W_j assigned to it. We define the multiplicity $m(l)$ of an edge tear set $ET(l)$ as the maximum number of times any of the node loops are

	Edges	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	
μ_i		5	2	3	1	5	3	1	1	1	Weights, W_j
0	$\upsilon(1)$	x	x					x			
0	$\upsilon(2)$		x	x			x				
0	$\upsilon(3)$			x		x					
0	$\upsilon(4)$	x	x		x				x		
0	$\upsilon(5)$	x							x	x	
LEV=∅	λ_j	3	3	2	1	1	1	1	2	1	
	η_j	0.6	1.5	0.67	1	0.2	0.33	1	2	1	
	ϵ_j	0	0	0	0	0	0	0	0	0	

Figure 9. Loop/edge incidence matrix for digraph in Figure 8a.

Edge	8	2	4	7	9	3	1	6	5
λ_j	2	3	1	1	1	2	3	1	1
W_j	1	2	1	1	1	3	5	3	5
b_j^{ϵ}	0	0	0	0	0	0	0	-	-
b_j^{W}	3	4	6	6-2/3	7-⅓	8	13	-	-

Figure 10. Multiplicity and weight bounds for reordered edges for Figure 8.

		e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9
μ_i		5	2	3	1	5	3	1		1
0	$\upsilon(1)$	x	x					x		
0	$\upsilon(2)$		x	x			x			
0	$\upsilon(3)$			x		x				
1	$\upsilon(4)$	x	x		x				x	
1	$\upsilon(5)$	x							x	x
	λ_j	1	2	2	0	1	1	1	0	0
	η_j	0.2	1	0.67	0	0.2	0.33	1	0	0
	ϵ_j	1	1	0	1	0	0	0	1	1

Figure 11. Loop/edge incidence matrix after tearing edge 8.

Edge	7	3	6	5	2	1	4	8	9
λ_j	1	2	1	1	2	1	0	0	0
W_j	1	3	3	5	2	5	1	1	1
ϵ_j	0	0	0	0	1	1	1	1	1
b_j^{ϵ}	0	0	1	1	1	1	1	1	1
b_j^{W}	5	7	6	8	8	-	-	-	-

Figure 12. Multiplicity and weight bounds for reordered edges for Figure 11.

torn. The weight of an edge tear set is the sum of the weights assigned to the edges in the edge tear set $ET(l)$. Our algorithm will locate an edge tear set such that no other edge tear set has a lower multiplicity, and, of all those with the same multiplicity, the selected set has the minimum weight. The algorithm, with explanation, is as follows.

I. For the irreducible digraph, locate all node loops (an algorithm which extends in an obvious and minor way the LOOPFINDER algorithm in Forder and Hutchison [1969] is recommended) and display them in a node loop incidence matrix, $\underline{\underline{M}}$.

II. Assign to each loop i a multiplicity $\mu_i = 0$, set level count $LEV = 0$, set $\epsilon_{best} = W_{best} = \infty$, set $\hat{n}$ = number of rows in $\underline{\underline{M}}$, set $WTSUM = 0$.

III. Assign to each edge e_j the following three numbers:

A. An untorn loop count λ_j, where λ_j is the number of loops which are as yet untorn and which include edge e_j.

B. An edge efficiency η_j, where $\eta_j = \lambda_j/W_j$. η_j equals the number of loops which would be torn per unit of assigned weight for the edge e_j.

C. An edge multiplicity ϵ_j, where

$$\epsilon_j = \max_k\{\mu_k | e_j \text{ appears in loop } \upsilon(k)\}$$

With these numbers we can assess the value of adding edge e_j next to the edge tear set partially completed. If edge e_j is added to the edge tear set, λ_j more loops will be torn with an efficiency per unit of edge weight η_j. At least one loop in the set of all loops will be torn with multiplicity $\epsilon_j + 1$.

IV. Increment LEV by one. Reorder onto a list, List I of level LEV, the indices of all edges in increasing order of multiplicity, reordering edges with equal multiplicity in order of decreasing efficiency.

V. For each edge e_i on the ordered List I, develop a lower bound on multiplicity. The lower bound assumes all edges before e_i on the ordered list are not in the edge tear set and that edge e_i is and all following can be in the edge tear set. To establish the multiplicity bound use e_i and, in order, each of the edges following. If $\hat{n}$ loops remain to be torn, then the fewest edges needed to tear all the remaining loops, using edges e_i, ... e_p would be such that

$$\lambda_i + \lambda_{i+1} + \ldots + \lambda_{p-1} + \theta_1\lambda_p = \hat{n} \qquad (1)$$

where $0 < \theta_1 \leq 1$. Since this number assumes loops cut by each edge are different from those cut by the other edges, we clearly have a lower bound on the number of edges needed from the sequence used. The multiplicity ϵ_p establishes the multiplicity bound b_i^{ϵ} for edge i. If an insufficient number of edges exist for (1) to be established for an edge, no multiplicity bound exists for that edge.

VI. Establish, for each edge e_i on the ordered List I for level LEV and for which a multiplicity bound exists, a lower bound on the required sum of weights to complete the edge tear set. For edge e_i, consider all edges following e_i up to e_q where q is the last edge with multiplicity $\mu_q = b_i^{\epsilon}$. Reorder these edges e_{i+1} to e_q in order of decreasing edge efficiency onto a temporary list, List II. Let this list contain the indices $S_1, S_2, \ldots . S_{q-i}$. Select e_i and just enough edges in order from List II such that

$$\lambda_i + \lambda_{S_1} + \ldots + \lambda_{S_{t-1}} + \theta_2\lambda_{S_t} = \hat{n} \qquad (2)$$

where

$$0 < \theta_2 \leq 1.$$

The lower bound on the sum of weights for edge e_i is then

$$b_i^W = W_i + W_{S_1} + W_{S_2}^u + \ldots + W_{S_{t-1}} + \theta_2 W_{S_t} + WTSUM$$

Note, the bounds b_i^{ϵ} for edges on List I for Level LEV are in increasing order but the bounds b_i^W may not be.

VII. Set $NXT(LEV) = 1$.

VIII. Set $k = NXT(LEV)$ and increment $NXT(LEV)$ by one. Set $\hat{k}$ = k-th index on List I for level LEV.

IX. A. If $b_{\hat{k}}^{\epsilon} >$ best or if $b_{\hat{k}}^{\epsilon}$ does not exist, go to Step XII.

B. If $b_{\hat{k}}^{W} \geq W_{best}$, return to step VIII.

X. Add edge $e_{\hat{k}}$ to the edge tear set as follows. It will be the LEV-th edge in the set.

A. Increment *WTSUM* by W_{ℓ}.

B. For each loop $v(i)$ in which e_{ℓ} appears, increment μ_i by one.

C. If μ_i just becomes one: 1. decrement $\hat{n}$ by one; and 2. for each edge e_u in loop $v(i)$, decrement the untorn loop counter by one.

XI. If all loops $v(i)$ are not yet torn (at least one $\mu_1 = 0$), return to Step IIIB. Otherwise,

A. Set $\epsilon_{best} = b_k^{\epsilon}$ and $W_{best} = b_k^{W}$ for current tear set.

B. Save current edge tear set as best.

C. Go to Step XIII.

XII. All edges not yet considered on List I at level *LEV* need not be considered further as they cannot lead to a better tear set than the best found so far.

A. Decrement *LEV* by one.

B. If *LEV* = 0, exit algorithm.

C. Set $k = NXT(LEV) - 1$, and set $k = k$-th index on List I for level *LEV*.

XIII. Remove edge e_{ℓ} from current edge tear set by

A. Decrement *WTSUM* by W_{ℓ}.

B. For each loop $v(i)$ in which e_{ℓ} appears, decrement μ_i by one.

C. If μ_i just becomes zero: 1. increment $\hat{n}$ by one; and 2. then for each edge e_n in loop $v(i)$, increment the untorn loop counter λ_u by one.

XIV. Return to Step VIII.

We can illustrate the algorithm with an example, the digraph of Figure 8a. We arbitrarily assign weights to the edges and use the algorithm to find a best edge tear set.

Step I. The five node loops for the digraph in Figure 8a are illustrated in the incidence matrix shown in Figure 9. Loop $v(1)$ contains edges e_1, e_2 and e_7. Weights W_j are assigned across the top to each edge, e.g., edge e_1 has a weight of 5.

Step II. Again looking at Figure 9, we see $\mu_i = 0$ assigned to each loop along the left border. $\hat{n}$ is 5 here.

Step III. Figure 9 also has λ_j, η_j and ϵ_j values assigned for each edge. $\lambda_1 = 3$ because edge e_1 appears in three as yet untorn loops. $\eta_1 = 3/5 = 0.6$ and $\epsilon_1 = \max(\mu_1, \mu_4, \mu_5) = 0$. Since all loops are as yet untorn, all loops torn in the first step will be torn once, i.e., with a multiplicity of $\epsilon_j + 1 = 1$.

Step IV. Increment *LEV* to one. List I will be the indices (8,2,4,7,9,3,1,6,5). Edge e_8 has the highest efficiency ($\eta_8 = 2$) and edge e_5 the lowest ($\eta_5 = 0.2$).

Step V. Figure 10 shows the multiplicity bounds for each edge with edges reordered as done in Step IV. For edge 8 we need to tear at least e_8 and e_2 to tear $\hat{n} = 5$ loops so $b_8^{\epsilon} = \max(\epsilon_8, \epsilon_2) = 0$. For edge e_2 we need to tear it and at least edges e_4 and e_7 to tear 5 loops so $b_2^{\epsilon} = \max(\epsilon_2, \epsilon_4, \epsilon_7) = 0$. $b_4^{\epsilon} = \max(\epsilon_4, \epsilon_7, \epsilon_9, \epsilon_3) = 0$, and so forth. Edge e_6 has no bound since e_6 and all edges following (e_5) can tear only 2 loops. Similarly, e_5 has no bound.

Step VI. In a manner similar to establishing the multiplicity bound for an edge, we establish a weight bound. Since all multiplicity bounds are equal to zero, list II is the same as list I. For the first edge, e_8, we need edge e_8 and e_2 at least to tear 5 loops so $b_8^W = W_8 + W_2 + WTSUM = 1 + 2 + 0 = 3$. $b_2^W = W_2 + W_4 + W_7 + WTSUM = 2 + 1 + 1 + 0 = 4$. For edge e_7 we need e_7, e_9, e_3 and only ⅓ of e_1 to get 5 loop tears so $b_7^W = W_7 + W_9 + W_3 + W_1/3 + WTSUM = 6\frac{2}{3}$.

Step VII. Set $NXT(1) = 1$.

Step VIII. Set $k = 1$, $NXT(1) = 2$ and $\hat{k} = 8$, the first index on List I for level 1.

Step IX. $b_8^{\epsilon} = 0 < E_{best} = \infty$ and $b_8^W = 3 < W_{best} = \infty$ so continue to step X.

Step X. We shall add edge e_8 as the first edge in the tear set.

WTSUM will be incremented from zero to zero $+ W_8 = 3$. μ_4 and μ_5 are incremented to unity because e_8 appears in loops $v(4)$ and $v(5)$. $\hat{n}$ will be decremented to $5 - 2 = 3$ since all loops which are

		e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9
μ_i	λ_1	5	2	3	1	5	3	1	1	1
1	$v(1)$	x	x					x		
0	$v(2)$		x	x			x			
0	$v(3)$			x		x				
1	$v(4)$	x	x		x				x	
1	$v(5)$	x							x	x
	λ_j	0	1	2	0	1	1	0	0	0
	η_j	0	0.5	0.67	0	0.2	0.33	0	0	0
	ϵ_j	1	1	0	1	0	0	1	1	1

Figure 13. Loop/edge incidence matrix after tearing edge 8 and then edge 7.

Edge	e_3	e_6	e_5	e_2
λ_j	2	1	1	1
W_j	3	3	5	2
ϵ_j	0	0	0	1
b_j^{ϵ}	0	0	1	-
b_j^{W}	5	10	9	-

Figure 14. Multiplicity and weight bounds for reordered edges for Figure 13.

		e_1	2	3	4	5	6	7	8	9
1	$v(1)$	x	x					x		
1	$v(2)$		x	x			x			
0	$v(3)$			x		x				
1	$v(4)$	x	x		x				x	
0	$v(5)$	x							x	x
	λ	1	0	1	0	1	0	0	1	1
	η	0.2	0	0.33	0	0.2	0	0	1	1
	ϵ_j	1	1	1	1	0	1	1	1	0

Figure 15. Loop/edge incidence matrix after tearing edge 2.

	e_9	e_5
λ	1	1
W	1	5
b_j^{ϵ}	0	1
b_j^{W}	8	-

Figure 16. Partially developed multiplicity and weight bounds for reordered edges for Figure 15.

torn are torn for the first time here. The untorn loop counter for e_1 is decremented by 2 since two loops in which it occurs have just been torn, e_2 by 1, e_3 by zero and so forth.

Step XI. For our example $\mu_1 = \mu_2 = \mu_3 = 0$ yet so we return to Step IIIB.

Steps IIIB and C, IV, V and VI lead to the results in Figures 11 and 12. *LEV* is reset to 2 in Step IV.

Step VII. $NXT(2) = 1$
Step VIII. $k = 1$, $NXT(2) = 2$. $k = 7$.
Step IX. Continue to Step X.
Step X. Add edge e_7 to the edge tear set.
Step IX will return us to IIIB where Steps IIIB and C, IV, V and VI lead to the results in Figures 13 and 14.

Step VII. $NXT(3) = 1$
Step VIII. $k = 1$, $NXT(3) = 2$, $k = 3$.
Step IX. Continue to Step X.
Step X. Add edge e_3 to the edge tear set. We now find all loops torn (no $\mu_i = 0$). So, we set $\epsilon_{\text{best}} = b_3^{\epsilon} = 0$ and $W_{\text{best}} = b_3^{W} = 5$.

Step XII. No further edges at level 3 need be considered. Decrement *LEV* to 2 and set $k = NXT(2) - 1 = 1$. Set $\hat{k} = 7$.

Step XIII. Remove edge e_7 from the tear set. Decrement μ_1 by one, set $\hat{n} = 3$, and increment λ_1, λ_2 and λ_7 by one. Note, we have simply recovered Figure 11 by these steps.

Step XIV. Go to Step VIII.

Step VIII. Set $k = NXT(2) = 2$, $NXT(2) = 3$ and $\hat{k} = 3$.

Step IX. See Figure 12. We are now looking at replacing e_7 by e_3 as the second tear. $b_3^{\epsilon} = 0 = \epsilon_{\text{best}} = 0$. $b_3^{W} = 7 \geq W_{\text{best}} = 5$ so we go to Step VIII. (It would require a tear weight of at least 7 so skip).

Step VIII. $k = NXT(2) = 3$, $NXT(2) = 4$ and $\hat{k} = 6$.

Step IX. $b_6^{\epsilon} = 1 > \epsilon_{\text{best}} = 0$ (see Figure 11). Putting edge 6 or any following in as a tear would raise the multiplicity of the solution to 2 (a loop would become doubly torn) so we can forget looking at level 2 options. Go to Step XII.

Step XI. Set *LEV* to 1 (we should now return to Figures 9 and 10), $k = NXT(1) - 1 = 1$, $\hat{k} = 8$.

Step XIII. Delete edge e_8 from the tear set.

Step VIII. Set $k = NXT(1) = 2$, $NXT(LEV) = 3$ and $\hat{k} = 2$.

Continuing (see Figure 10), we find replacing e_8 by e_2 will give a weight bound of 4 which is less than 5, the best so far. We develop the loop incidence matrix and bounds in Figures 15 and 16 and find b_9^{W} at level 2 is already up to a minimum tear weight of 8 so we stop looking with e_2 as the first level tear. The next first level tear option is e_4 (Figure 10), but it has a weight bound of 6 so we can stop altogether. The best tear set is e_8, e_7 and e_1 with a multiplicity of 1 (an exclusive tear set) and a tear weight of 5. Note we examined only alternatives e_3 and e_6 (no effort required) at level 2, returned to level 1 and followed one more false trail based on e_2, e_9. Very few options had to be explored to find the best.

NOTATION

b_j^{ϵ} = multiplicity bound for edge j in the branch and bound algorithm
b_j^{W} = weight bound for edge j in the branch and bound algorithm
e_j = edge j in a digraph
E = set of edges in a digraph
$ET(k)$ = k-th edge tear set for a digraph
$G(N,E)$ = a digraph of the set N of nodes and the set E of directed edges connecting them
LEV = level index in the branch and bound algorithm. When *LEV* equals one, the algorithm is looking for the first edge to tear, it equals two when looking for the second edge, etc.
$m(k)$ = multiplicity of edge tear set $ET(k)$; i.e., the maximum number of times any loop in the digraph is torn by the edge tear set
M = node loop incidence matrix; rows correspond to the node loops, columns to edges.
n_k = node k in a digraph
$\hat{n}$ = the number of rows in the node incidence matrix M
N = the set of nodes in a digraph
$NXT(LEV)$ = a list of pointers, one for each level. $NXT(LEV)$ points to the next edge to be examined if the branch and bound algorithm returns to level *LEV*.
$p(k)$ = path k in a digraph
S_k = set of edge indices used in branch and bound algorithm corresponding to a reordering of the edges in order of decreasing edge efficiency, η_j
W_j = user assigned weight for edge e_j
W_{best} = the lowest weight sum found so far for any completed edge tear set
$WTSUM$ = the sum of the edge weights corresponding to the tear edges already selected in the lower levels

Greek Letters

ϵ_{best} = lowest multiplicity found so far for any edge tear set
ϵ_j = multiplicity resulting when selecting edge e_j next in the branch and bound algorithm
η_j = edge efficiency for selecting edge e_j next. Edge efficiency is defined as the ratio λ_j/W_j
θ_1 = defined in Step V of the branch and bound algorithm. A fraction of an untorn loop count needed to make Eq. 1 an equality.
θ_2 = defined in Step VI of the branch and bound algorithm. A fraction of an untorn loop count needed to make Eq. 2 an equality.
λ_j = untorn loop count for edge e_j. The untorn loop count is the number of loops which are as yet untorn and which include edge e_j.
μ_i = loop multiplicity in the branch and bound algorithm. Indicates the number of times loop $v(i)$ has been torn so far for the edges selected to be in the current edge tear set.
$v(l)$ = a node loop l in the digraph $G(N,E)$

ACKNOWLEDGMENT

This work was sponsored in part by NSF Grant No. CPE-7801809.

LITERATURE CITED

Barkley, R. W., and R. L. Motard, "Decomposition of Nets," *Chem. Eng. J.*, **3**, 265 (1972).
Cavett, R. H., "Application of Numerical Methods to the Convergence of Simulated Processes Involving Recycle Loops," Am. Pet. Inst. Repr., No. 04-63 (1963).
Christensen, J. H., and D. F. Rudd, "Structuring Design Computations," *AIChE J.*, **15**, 94-100 (1969).
CONCEPT Mark III User Manual, CAD Centre, Cambridge, England (1973).
Forder, G. J., and H. P. Hutchison, "The Analysis of Chemical Plant Flowsheets," *Chem. Eng. Sci.*, **24**, 771-85 (1969).
Motard, R. L., M. Shacham, and E. M. Rosen, "Steady-State Chemical Process Simulation," *AIChE J.*, **21**, 417-436 (1975).
Newman, D. J., and L. A. Klein, "Recent Developments in Nitric Acid Manufacture," *Chem. Eng. Prog.*, **68**(4), 62 (1972).
PACER 245 User Manual, Digital Systems Corp., Hanover, New Hampshire (1971).
Rosen, E. M., and A. C. Pauls, "Computer Aided Chemical Process Design," *Comput. Chem. Eng.*, **1**, 11 (1977).
Seader, J. D., W. D. Seider, and A. C. Pauls, "FLOWTRAN Simulation, An Introduction," CACHE Committee, Ulrich's Books, Ann Arbor, Michigan (1974).
Upadhye, R. S., and E. A. Grens, "Selection of Decompositions for Process Simulation," *AIChE J.*, **21**, 136 (1975).

Manuscript received January 29, 1980; revision received October 16, and accepted October 24, 1980.