Structuring Design Computations

J. H. CHRISTENSEN and D. F. RUDD

University of Wisconsin, Madison, Wisconsin

Process design computations are represented by directed graphs whose edges correspond to streams of information flow between computational units. Algorithm I-R extends existing algorithms for finding the minimal sized blocks of units between which no recycle exists. Algorithm II-R orders the sequence of unit computations within a block to minimize the number of recycle parameters. Algorithm III-R uses the concept of indexing to order computations which evade algorithm II-R. This work is directed toward the evolution of efficient programs for computer-aided process design.

One approach to the formulation and optimization of designs for large processing systems involves the development of master or executive computer programs to coordinate subroutines which perform the design computations for the unit process modules. One of the earliest of these routines was CHEOPS, [Chemical Engineering Optimization System (1)] and a second example is PACER [Process Assembly Case Evaluator Routine (2)]. The use of executive design programs is now commonplace in industry.

We must remark that the state of these programs are such that only process analysis and optimization computations have yielded to the computer to any useful degree, and that computer-aided design synthesis programs remain in a primitive stage of development (3, 4).

In this report we examine methods for improving the efficiency of design executive routines, by reducing complex design problems to an elemental form. Some of the principles employed here have been discussed before (5 to 7) and this work represents an expansion of those principles as well as an examination of some new concepts.

RECYCLE CALCULATIONS

The solution of the design equations for a complex process often cannot begin unless values are assumed for certain of the variables, the recycle parameters. Process computations then proceed until new values of the recycle parameters are computed, and iteration is then performed to force agreement to some specified tolerance of the assumed and computed values of the recycle parameters (8).

The mathematical problem thus generated may be formulated as finding a solution to the set of equations

$$x = g(x) \tag{1}$$

where x is the k dimensional set of recycle parameters, and g is the set of new values of the recycle parameters generated by the solution of the design equations. There is strong reason to believe that the computational sequence which results in the fewest recycle parameters k is a computational sequence which leads more easily to the solution of the design equations. While there are exceptions to this rule, as cited near the end of this report, the premise of minimum recycle offers a good starting point for an unsolved design problem.

PARTITIONING AND ORDERING

The first efforts to reduce the labor of complex recycle calculations were devoted to partitioning the design subroutines into blocks such that all recycle computations occurred within the blocks, and ordering the blocks so that completing the computations in one block assured that all the variables needed to perform the computations in the next block would be known. Keeping the number of units in each block as small as possible would then reduce the time required to perform the process computations for the entire block (9, 10).

Mathematically, the flow diagrams or functional diagrams which are used to represent the relationships between computational units are directed graphs. These graphs contain nodes representing the units and edges representing the streams. Thus, there will be an edge directed from node A to node B with a weight w_{AB} if there are w_{AB} variables computed in unit A which are required for the computations in unit B, as shown in Figure 1. The convention will be that if no number is written next to edge (A, B), then $w_{AB} = 1$.

A cycle in a directed graph is a path following the directed edges which returns to its starting node. A directed graph is cyclic if it contains at least one cycle, and acyclic if it contains no cycles. The partitioning problem may then be defined as follows. Divide the directed graph of the process into blocks or subgraphs which are as small as possible such that the graph consisting of the blocks and edges between them is acyclic.

Sargent and Westerberg (5) perform partitioning directly with a list type of representation of the computational process flow diagram. They trace a path backwards along the links connecting the units until a loop is found; the units in this loop are then merged into a single unit representing the block containing the loop, and the trace is then continued until all loops are partitioned into blocks. In the next sections we expand on this idea.

ALGORITHM I-R FOR PARTITIONING AND ORDERING

An acyclic graph with a finite number of nodes must have at least one node with no outgoing edges to any other node in the graph, because every path following the directed edges must terminate. The computations for the unit corresponding to such a node may follow all those for other nodes in the graph. If the node and its input edges are deleted from the graph, the remaining subgraph of the original acyclic graph must also be acyclic. Thus, the deletion process may be repeated on an acyclic graph until no nodes remain, and computation of the units may proceed in the reverse of the order in which the nodes were deleted. For example, in Figure 2 deletion may proceed in the order (E,D,C,B,A) and process com-

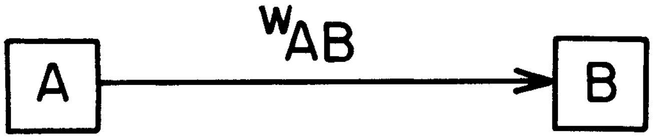


Fig. 1. Notation of directed graphs.

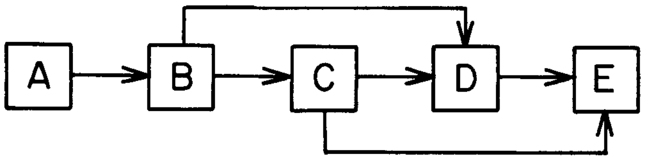


Fig. 2. An acyclic graph.

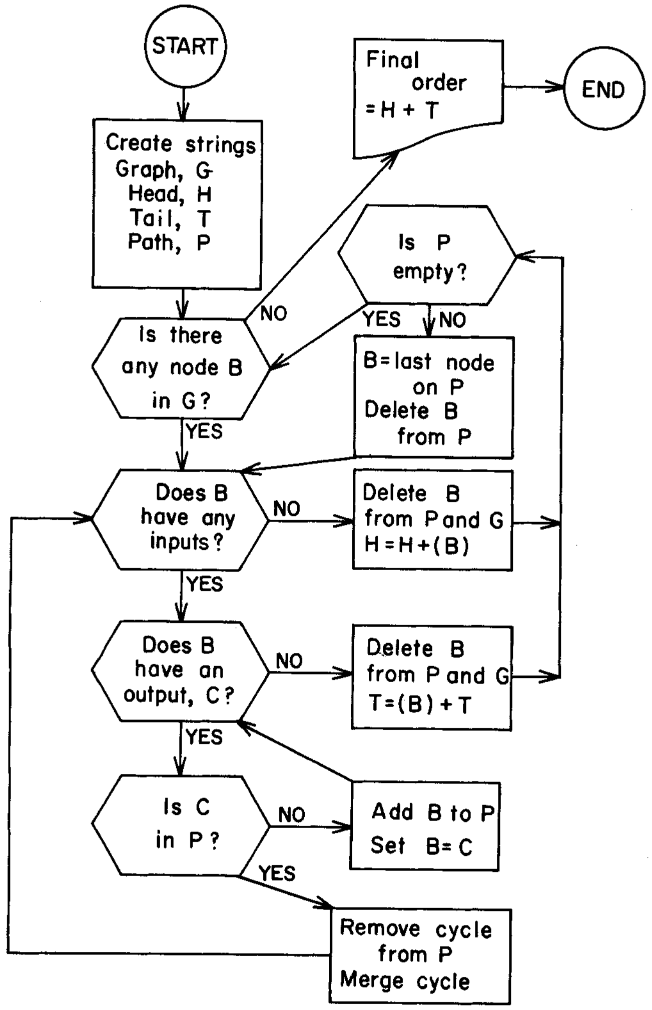

START
Create strings Graph, G Head, H Tail, T Path, P
Is there any node B in G?
Final order = H + T
END
Is P empty?
YES
NO
B = last node on P Delete B from P
Does B have any inputs?
Delete B from P and G H = H + (B)
Does B have an output, C?
Delete B from P and G T = (B) + T
Is C in P?
Add B to P Set B = C
Remove cycle from P Merge cycle

Fig. 3. Flow chart of Algorithm I-R.

putations in the order (A,B,C,D,E).

If the deletion process terminates with some nodes left in the graph, then it must be cyclic. The number of nodes to be partitioned may sometimes be further reduced by deleting from the graph any node with no input edges. This process may be repeated until no such nodes remain, and computation of the corresponding units may proceed in the same order as that in which the nodes were deleted.

Since any path traced backwards in a finite acyclic graph must terminate, there must be at least one node with no inputs in such a graph. Thus, the deletion process on such nodes will also produce an ordering of all the nodes of an acyclic graph. Furthermore, the successive portions of any graph which are ordered by either of the above two deletion procedures will not have any recycle between them.

All the nodes of a given cycle must necessarily be contained in the same block, as must the nodes of any other cycle which passes through portions of this cycle. Therefore, any cycle may be replaced, for the purpose of block partitioning, by a single node whose inputs and outputs are the same as those of the cycle. Combining a procedure for detecting and merging cycles with the above deletion procedures gives Algorithm I-R for partitioning and ordering, shown in Figure 3.

This algorithm differs from Sargent and Westerberg's Algorithm I, in that paths are traced forward as well as backward, and nodes may be added to either head or tail strings, which increases the speed of ordering. A partition and ordering of the first example of Sargent and Westerberg, with the graph shown in Figure 4, might be $(A)(B)(F)(G,H)(C)(M)(D,E,I,L)(J)(K)$, where each block is enclosed in separate parentheses.

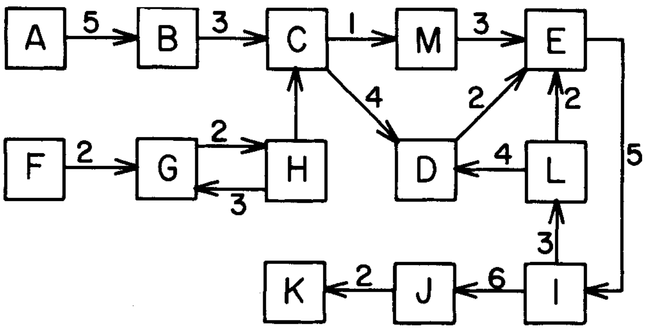


Fig. 4. Graph of block ordering example.

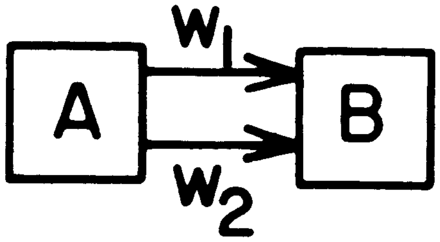


Fig. 5*a*. Portion of a graph containing parallel edges.

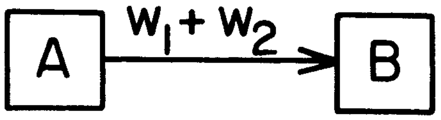


Fig. 5*b*. Merging of parallel edges.

MINIMUM RECYCLE

If none of the blocks found by Algorithm I-R contains more than one node, then since the graph of the blocks is acyclic, there need be no recycle computations at all. If this is not the case, then recycle computations will have to be performed within those blocks containing two or more nodes. It is then desirable to order the sequence $\{U\}$ of unit computations within each such block so as to minimize the number, k, of recycle parameters. This sequence is denoted by $\{U\}^*$ and the minimum number of recycle parameters by k^*.

One way of finding a sequence $\{U\}^*$ would be to delete from the block trial combinations of recycle streams having successively greater k, and testing for acyclicity with Algorithm I-R at each trial. When a combination is

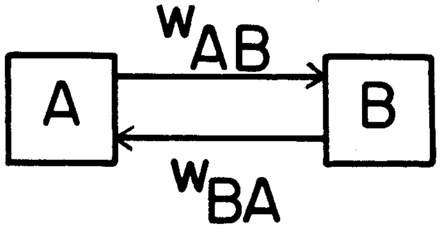


Fig. 6*a*. Portion of a graph containing two-way edges.

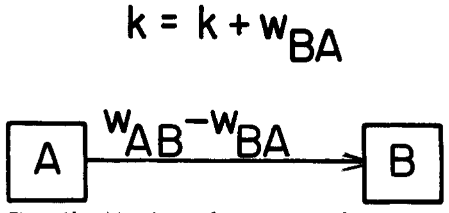


Fig. 6*b*. Merging of two-way edges when $w_{AB} > w_{BA}$.

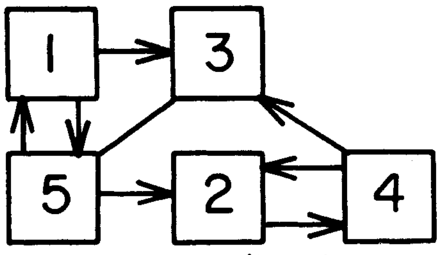


Fig. 7a. Graph of a block.

$k = 1 + 1 = 2$

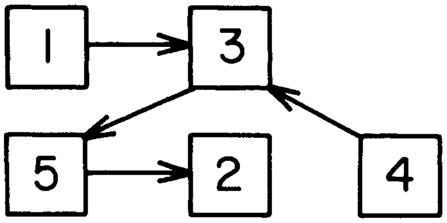


Fig. 7b. Acyclic graph produced by merging two-way edges.

found for which Algorithm I-R produces all sub-blocks with only one unit each, this sequence of sub-blocks would be $\{U\}^*$. For example, in the block (D,E,I,L) of the previous example, deletion of either one of the single edges having two parameters each still yields a cyclic sub-block, but deletion of the single edge having three parameters, that is, edge (I,L) yields the sub-block sequence $\{U\}^* = (L)(D)(E)(I)$ with $k^* = 3$ recycle parameters.

ALGORITHM II-R FOR MERGING EDGES AND NODES

For blocks of moderate size, the above procedure may require the examination of so many combinations of edges that the cost of finding an optimal sequence exceeds the saving in recycle computations. Sargent and Westerberg reduce the number of combinations by merging edges and nodes. First, if two nodes are connected by parallel edges as in Figure 5a, the graph cannot possibly be made acyclic by the deletion of one such edge unless all are deleted. Thus, these parallel edges are equivalent to a single edge containing all the parameters, as shown in Figure 5b. This combination process may be repeated until no parallel edges remain in the graph.

Secondly, Sargent and Westerberg note that two-way links as in Figure 6a may be replaced by net unidirectional links as in Figure 6b. This is justified because, if A precedes B in $\{U\}$, then the edge (B,A) must be a recycle stream, adding w_{BA} recycle parameters to the total. Similarly, if B precedes A, w_{AB} parameters must be added. When $w_{AB} > w_{BA}$, this is equivalent to the system obtained by deleting edge (B,A), adding w_{BA} as a bias to the number of recycle parameters k, and subtracting w_{BA} from w_{AB}. The opposite rearrangement would be made if $w_{AB} < w_{BA}$, adding w_{AB} to k; if $w_{AB} = w_{BA}$, then w_{AB} parameters must be added no matter which node comes first, and both edges may be deleted.

Merging of edges is sometimes sufficient to produce an acyclic graph, as in the example of Lee and Rudd (6) shown in Figure 7a. Merging of two-way edges yields the acyclic graph of Figure 7b, from which Algorithm I-R produces the sequence $\{U\}^* = (1)(4)(3)(5)(2)$ with $k^* = 2$. Similarly, merging of two-way edges in the block (G,H) of Figure 4 gives $\{U\}^* = (H)(G)$ with $k^* = 2$.

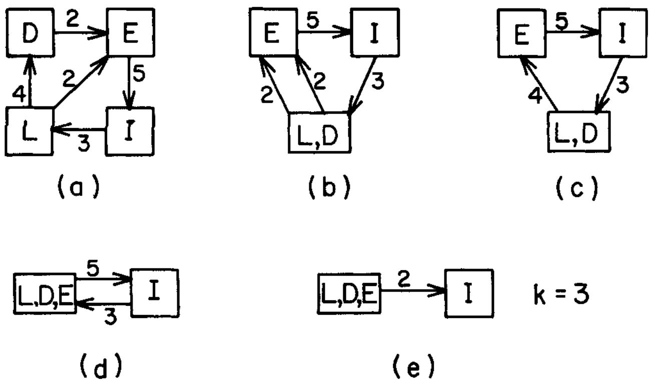


Fig. 8. Simplification of a block by merging.

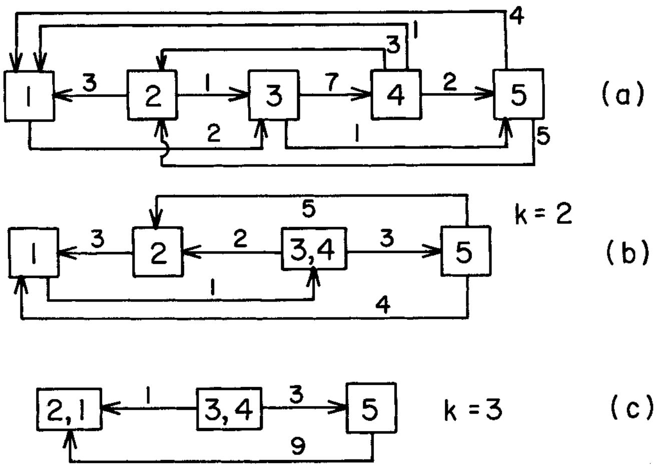


Fig. 9. Simplification by merging of a block with no simply linked units.

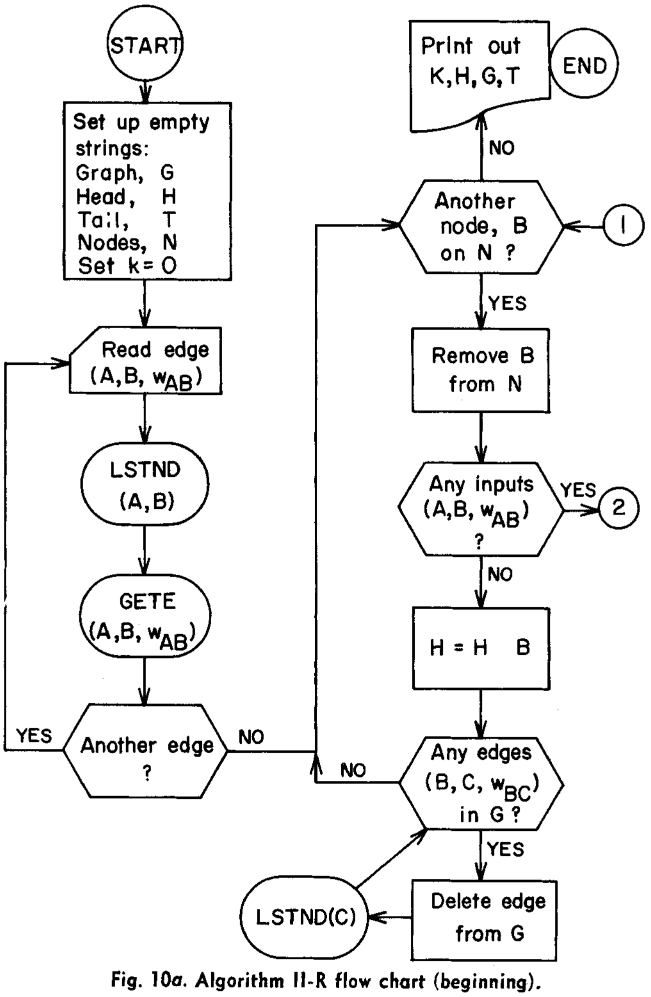
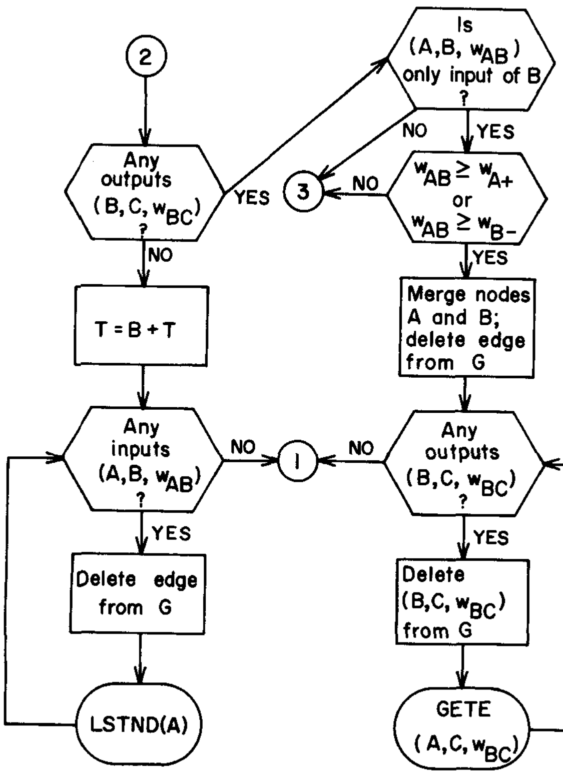


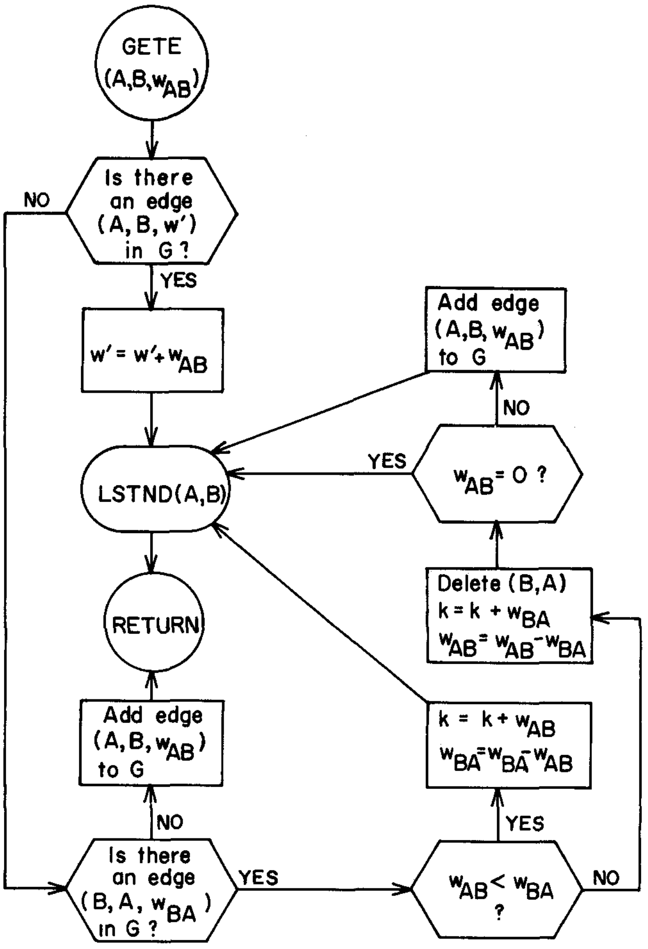
Fig. 10a. Algorithm II-R flow chart (beginning).



2
Any outputs (B, C, w_{BC}) ?
YES
NO
Is (A,B, w_{AB}) only input of B ?
NO
YES
3
NO
$w_{AB} \geq w_{A+}$ or $w_{AB} \geq w_{B-}$
YES
Merge nodes A and B; delete edge from G
T = B + T
Any inputs (A,B, w_{AB}) ?
NO
1
NO
Any outputs (B,C, w_{BC}) ?
YES
Delete edge from G
YES
Delete (B,C, w_{BC}) from G
LSTND(A)
GETE (A,C, w_{BC})

Fig. 10*b*. Flow chart of Algorithm II-R (continued).

The combinatorial problem may very often be further reduced by eliminating ineligible edges. If node A has w_{A+} total input parameters, and node B has w_{B-} total

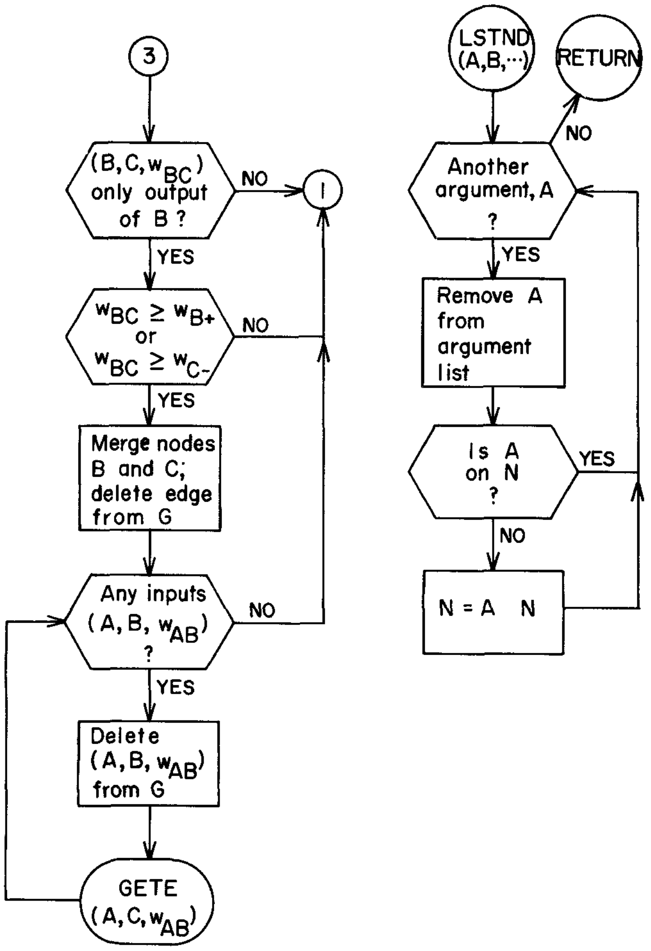


GETE (A,B,w_{AB})
NO
Is there an edge (A, B, w') in G ?
YES
w' = w' + w_{AB}
Add edge (A,B, w_{AB}) to G
NO
YES
LSTND(A,B)
w_{AB} = 0 ?
RETURN
Delete (B,A) k = k + w_{BA} $w_{AB} = w_{AB} - w_{BA}$
Add edge (A, B, w_{AB}) to G
k = k + w_{AB} $w_{BA} = w_{BA} - w_{AB}$
NO
YES
Is there an edge (B, A, w_{BA}) in G ?
YES
$w_{AB} < w_{BA}$?
NO

Fig. 10*d*. Flow chart of subroutine GETE(A, B, w_{AB}) for Algorithm II-R.

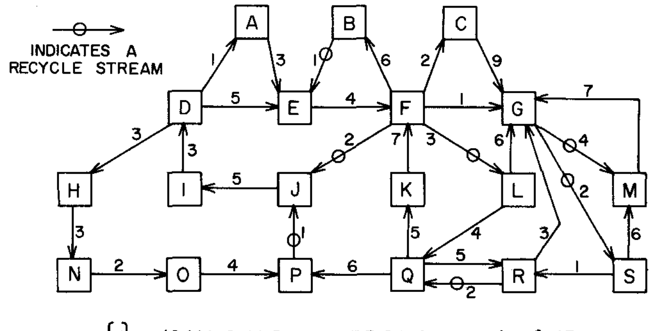
output parameters, then edge (A,B) need never be considered as a recycle stream in $\{U\}^*$ if $w_{AB} \geqq w_{A+}$ or $w_{AB} \geqq w_{B-}$. In the first case, some combination of the input streams of A, and in the second, of the output streams of B, could be used as recycle streams with the same effect as (A,B) and with no greater number of recycle parameters. If such an edge is the only output of node A or the only input to node B, then nodes A and B may be merged so that unit B and all its outputs will follow unit A and all its inputs in $\{U\}^*$.

Merging of nodes may generate new cases of edges to be merged, and vice versa, thus giving further reductions in the complexity of the block. As an example of this, consider the block (D,E,I,L) of Figure 4, redrawn separately in Figure 8*a*. Since $w_{LD} = 4 > w_{D-} = 2$, nodes L and D may be merged along the single input edge of D, giving the graph of Figure 8*b*. Merging of parallel edges gives the graph of Figure 8*c*; then nodes E and L,D may be merged along the single input to E, yielding



3
(B,C,w_{BC}) only output of B ?
NO
1
YES
$w_{BC} \geq w_{B+}$ or $w_{BC} \geq w_{C-}$
NO
YES
Merge nodes B and C; delete edge from G
Any inputs (A, B, w_{AB}) ?
NO
YES
Delete (A,B, w_{AB}) from G
GETE (A,C,w_{AB})
LSTND (A,B,···)
RETURN
NO
Another argument, A ?
YES
Remove A from argument list
Is A on N ?
YES
NO
N = A N

Fig. 10c. Algorithm II-R flow chart (concluded) and subroutine LSTND($A, B, \ldots$) flow chart.



$\{U\}^* =$ (S,M,L,Q,J,I,D,A,E,K,F,B,C,R,G,H,N,O,P); $K^* = 15$

Fig. 11. The example of Sargent and Westerberg.

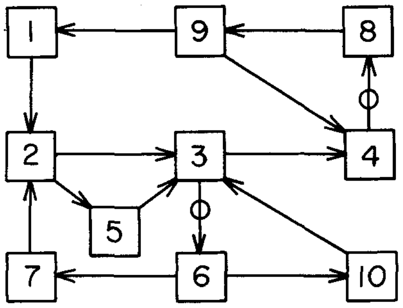


Fig. 12. The counter-example of Rubin.

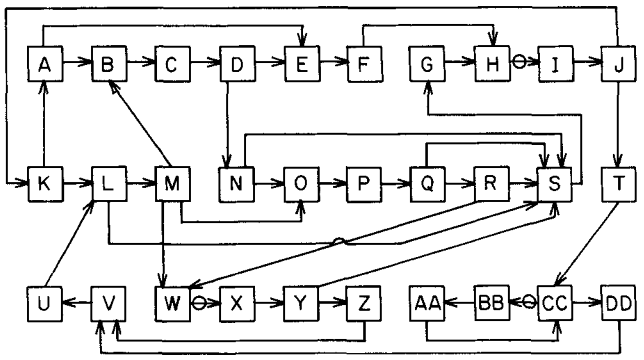
the graph of Figure 8*d*. Finally, merging the two-way edges thus generated gives the graph in Fig. 8*e* with $k = 3$. Algorithm I-R can now order this graph in the sequence $(L,D,E)(I)$, which is exactly the same optimal sequence as previously obtained; however, this time no combinatorial search was required.

The concept of ineligible edges is a generalization of Sargent and Westerberg's simply linked unit, that is, a node with both a single input and a single output. It is apparent that such a node can always be merged, since one of the edges will be ineligible. Thus, the use of ineligible edges can always produce an ordering whenever the use of simply linked units will, and can often produce an optimal sequence of units in a block having no simply linked units at all. As an example, consider the system of Rubin (*10*) cited by Lee and Rudd (*6*) and shown in Figure 9*a*. Here nodes 3 and 4 may be merged, with subsequent merging of edges giving Figure 9*b*. Merging of units 2 and 1 plus edge merging gives Figure 9*c*, which can be ordered by Algorithm I-R to give the optimal sequence 3,4,5,2,1 with $k^* = 3$.

Algorithm II-R, with the flow chart shown in Figure 10, applies edge and node merging systematically to simplify the structure of a block, and orders the units as far as possible into head and tail strings. It utilizes the interaction between edge and node merging to simplify the search for ineligible edges along which units may be merged. This algorithm is often sufficient to produce an optimal ordering of all the units in very complex blocks, such as the example in Figure 11 used by Sargent and Westerberg, the counter-example to Rubin's method shown in Figure 12, and the two examples of actual chemical process computations given in Figures 13 and 14.

ORDERING WHEN ALGORITHM II-R FAILS

Algorithm II-R has the advantages of speed, simplicity, and guaranteed optimality of any ordering which it produces; however, there are cases in which it will fail to produce an ordering. Nonetheless, in these cases the concept of ineligible edges is still very useful.



{U}* = (BB,AA,I,J,K,A,X,Y,T,CC,Z,DD,V,U,L,M,B,C,D,N,O,P,Q,R,W,E,F,S,G,H)
k* = 3

Fig. 14. Process computation example no. 2.

Consider the graph shown in Figure 15*a*, which is that of Figure 9 with all edge weights set equal to unity. There is then no single input or output edge which is ineligible; hence, Algorithm II-R fails. It is apparent, however, that only node 3 or node, 4 all of whose inputs are eligible to be recycle streams, can possibly be the first node in $\{U\}^*$. Furthermore, if node 3 comes first, then at least $w_{3+} = 2$ recycle parameters will be required; or if node 4 is first, at least $w_{4+} = 1$ recycle parameter will be required. The best course to try would thus be to put node 4 into the head, delete the node from the graph, and set $k = w_{4+} = 1$. Algorithm II-R can now produce an optimal ordering of the remaining graph shown in Figure 15*b*; hence, the original graph now has the order (4,5, 2,1,3) with $k = 2$. That this is the optimal ordering is easily seen by the fact that the only other node which can possibly be first in $\{U\}^*$, i.e., node 3, requires at least two recycle parameters. Thus, an optimal ordering has been produced by examining only one of 55 possible combinations of two or less recycle streams.

Another optimal ordering could be produced by noting that only node 1 with $w_{1-} = 1$ or node 3 with $w_{3-} = 2$ could possibly be at the right end of $\{U\}^*$, since all the outputs of the last node must be recycle streams and hence all must be eligible. If node 1 is deleted and put into the tail, Algorithm II-R orders the remaining graph to give $\{U\}^* = (3,4,5,2,1)$ with $k^* = 2$.

USING INDEX NODES: ALGORITHM III-R

The above observations may be generalized by defining an index node as one all of whose inputs, or outputs, or both, are eligible to be recycle streams. This terminology is chosen because such a node points to an allowable combination of recycle streams. If more than one index node is required to produce an ordering, then optimality

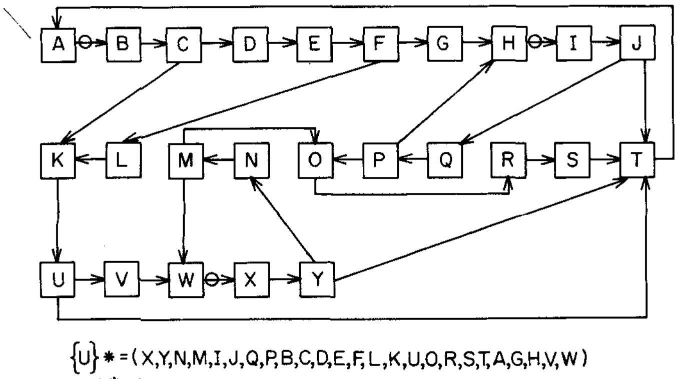


Fig. 13. Process computation example no. 1.

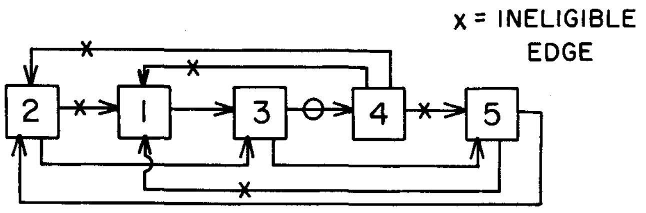


Fig. 15*a*. A graph which Algorithm II-R cannot order.

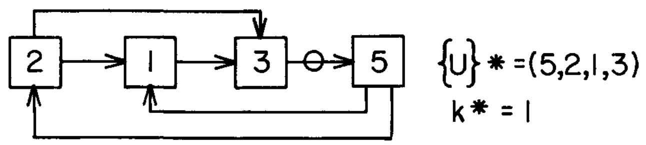
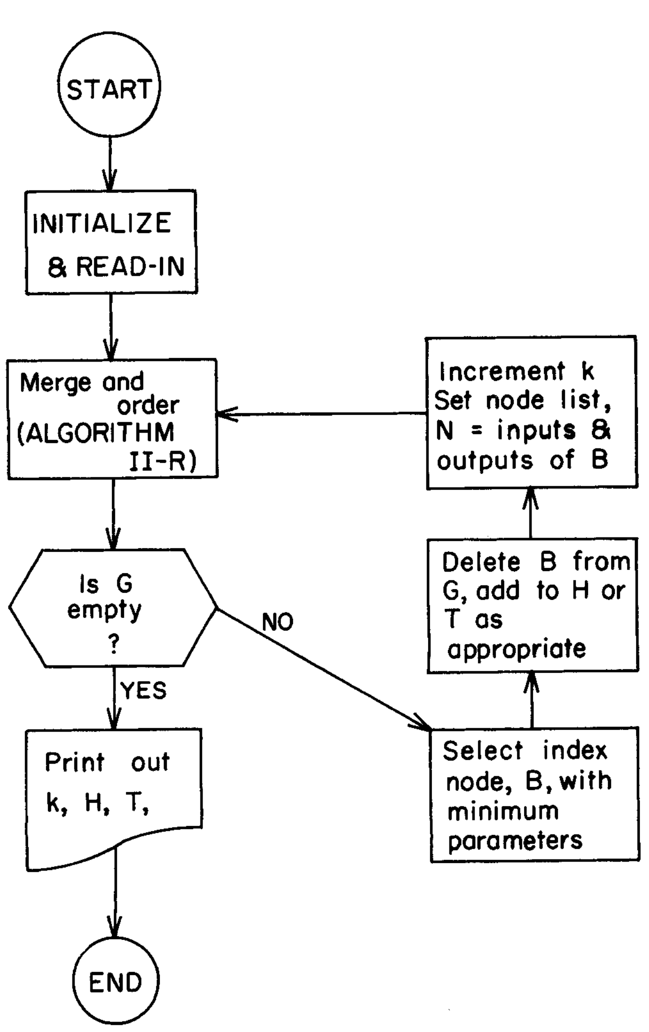


Fig. 15*b*. Optimal order obtained by deleting index node 4.



START
INITIALIZE & READ-IN
Merge and order (ALGORITHM II-R)
Is G empty ?
NO
YES
Print out k, H, T,
END
Increment k Set node list, N = inputs & outputs of B
Delete B from G, add to H or T as appropriate
Select index node, B, with minimum parameters

Fig. 16. Flow chart of Algorithm III-R.

can only be guaranteed by examining permutations of index nodes; such a procedure could quickly become overwhelmingly cumbersome.

A straightforward procedure would be to alternate Algorithm II-R with deletion of index nodes until an ordering is finally obtained. A close, and very often exact, approximation to optimal ordering can be attained by deleting each time one index node which gives the minimum increase in the number of recycle parameters. This procedure, shown in Figure 16, is Algorithm III-R.

DISADVANTAGES OF PROCESS UNIT SUBROUTINES

The great disadvantage of unit subroutines is that inputs and outputs of a given unit computation are fixed at the time of writing the routine. This fixed direction of information flow may be opposite to that necessary for efficient computation of a given process; special routines may then be required to force information flow in the opposite direction.

A further disadvantage is that fixing information flow in the direction of material flow may cause calculations within the routine to be more complicated than necessary. Consider, for example, the steady state material balances for the nth stage in a series of countercurrent stirred-tank reactors with the second-order reaction $A + B \xrightarrow{k_1} C$ (*11*):

$$xA_{n-1} - (1-x)A_{n+1} - A_n - k_1\theta A_n B_n = 0$$
$$xB_{n-1} - (1-x)B_{n+1} - B_n - k_1\theta A_n B_n = 0 \qquad (2)$$

Here x represents the fraction of outlet volume flow L_n which goes from stage n to stage $n + 1$; A_n and B_n represent the molar concentrations of A and B, respectively, in stage n; k_1 is the reaction rate constant; and θ is the reactor residence time. In this example, L_n, x, and $k_1\theta$ are assumed to be the same for all stages. If information flow in a subroutine representing this stage follows the direction of material flow, then a quadratic equation must be solved to get A_n and B_n in terms of the stage inputs. This increases the time required for computation in the unit by an order of magnitude over that required if one of the material input streams were a computational output.

A final disadvantage is that information about process variables is often not contained in neatly defined streams; it may then be necessary to add a large number of units and streams to the structure whose sole purpose is to force the problem into a proper unit and stream format. Thus, in the present example, a splitter unit must be added at each stage, thus doubling the total number of units, as shown in Figure 17 for $N = 4$ stages.

INTELLIGENT USE OF RECYCLE ORDERING

Although partitioning and ordering of units for minimum recycle parameters may produce computational sequences which are both inefficient and unstable, it can provide insight into the structure of the process which may yield efficient computational procedures. For instance, in the above example, Algorithm II-R can produce a minimum recycle ordering such as that shown in Figure 17. A similar ordering can be produced for any number of stages N, giving $N/2$ recycle streams (dropping the remainder for odd N). However if a computational scheme can be found which does not require solving a quadratic equation at each stage, saving in computation time might be accomplished.

Such a scheme is to rearrange Equations (2) to give:

$$A_n = (xA_{n-1} + (1-x)A_{n-1})/(1 + k_1\theta B_n)$$
$$B_n = (xB_{n-1} + (1-x)B_{n-1})/(1 + k_1\theta A_n) \qquad (3)$$

Initial values are assumed for all A_n and B_n, $n = 1,2 \ldots.,$ N; Equations (3) are applied to get new values for A_n and B_n for each stage in order; and the sequence is repeated with the new values until convergence is obtained. Note that by doubling the number of recycle parameters, total computation time per iteration has been considerably reduced.

This iteration scheme is very stable, giving satisfac-

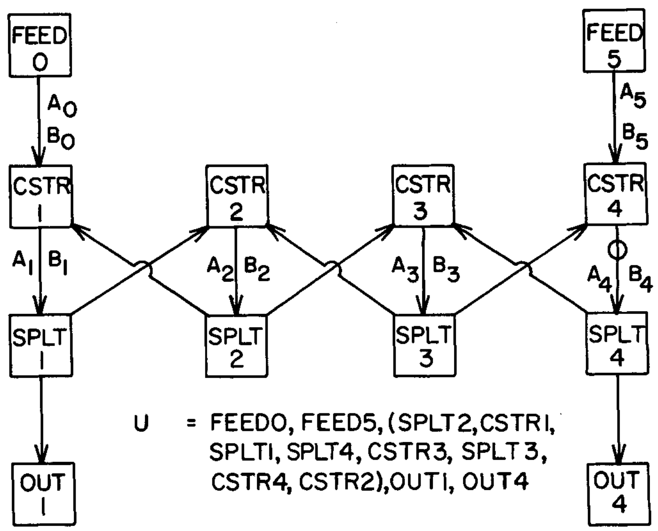


Fig. 17. Minimum recycle ordering of a system of four countercurrent stirred-tank reactors.

tory convergence in 25 iterations for N up to 10 stages with initial guesses far removed from the correct values. The values obtained agree with those given by Frank (*11*) for N = 2,3,5,8, and 10 stages with $A_0 = B_{N-1} = 1.0$, $A_{N-1} = B_0 = 0$, $k_1\theta = 4$, and initial guesses $A_n^{(0)} = B_n^{(0)} = 0.5$ for $n = 1, \ldots, N$. This method is much more stable than the iterative methods discussed by Frank. Also it is much simpler and faster than the technique finally adopted by Frank of solving the transient differential equations of the system numerically in order to obtain the steady state values. The advantages relative to this last method can be expected to increase with the complexity of the reaction system.

This example clearly indicates that the development of efficient computational schemes for process design often depends on the direct examination of the equations describing individual process units. The next paper of this series presents a systematic means of representing and utilizing the structural properties of the system of equations associated with optimizing a process design.

LITERATURE CITED

1. Hughes, R. R., E. Singer, and M. Souders, *Proc. Sixth World Petrol. Cong.*, Frankfurt (1963).
2. Shannon, P. T., et al., *Chem. Eng. Progr.*, **62**, 6 (1966).
3. Rudd, D. F., *AIChE J.*, **14**, 2 (1968).
4. Masso, A. H., and D. F. Rudd, *ibid.*, **15**, 9 (1969).
5. Sargent, R. W. H., and A. W. Westerberg, *Trans. Ind. Chem. Eng.*, **42**, 190 (1964).
6. Lee, W., and D. F. Rudd, *AIChE J.*, **12**, 6 (1966).
7. ———, J. H. Christensen, and D. F. Rudd, *AIChE J.*, **12**, 6 (1966).
8. Cavett, R. H., *API Division of Refining*, Philadelphia (May 1963).
9. Ravicz, A. E., and R. L. Norman, *Chem. Eng. Progr.*, **60**, 5, 71 (May, 1964).
10. Rubin, D. I., *Chem. Eng. Progr. Symposium Ser. No. 37*, **58**, 54 (1962).
11. Frank, A. Preprint 1E, AIChE 59th National Meeting, Columbus, Ohio (May 1966).

Manuscript received August 10, 1967; revision received January 16, 1968; paper accepted January 18, 1968.

The Estimation of Parameters for a Commonly Used Stochastic Model

A. L. SWEET and J. L. BOGDANOFF
Purdue University, Lafayette, Indiana

A region into which particles arrive in a random manner, remain a random amount of time, and then leave is considered. This model is used in penetration theories of heat and mass transfer. From observations of the number of particles present at any time, it is desired to estimate arrival and exit statistics, residence time statistics, and average rates of transfer across the region. Assuming arrival is a Poisson process, equations governing the above statistics are derived. Some problems in spectral analysis arising from the use of nondifferentiable stochastic processes are solved. Estimators for important parameters are discussed, and it is shown that generally they are biased. A derivation linking the rate of transfer across the region with the rates of transfer of particles is obtained and compared with other such results.

Consider a region into which particles arrive in a random manner, remain a random amount of time, and then leave. This paper is concerned with the particular problem of estimating various physical properties of interest exclusively from counts of the number of particles present in the region at any time. This problem appeared in the literature in 1916 (*14*) in connection with investigations of colloid statistics, and again in 1953 (*1*) in connection with estimating the speed of organisms. In 1954, under the assumption that the arrival process is Poisson, general results concerning properties of the number of particles in the region were derived (*2*). Recently, in application to heat transfer in fluidized beds (*3*), a set of assumptions about the physical nature of the arrival process led to results which, in part, were identical to those obtained (*2*). Because the previous work has been done in widely differing fields and on different aspects of this problem, and also because of a number of unusual features connected with the estimation problem, the authors felt further discussion of the problem to be worthwhile. In particular, this paper discusses the estimation of the following quantities: entrance and exit statistics of the particles, residence times of the particles, and transfer properties of the region, in order to show that consistent estimators of the above properties are biased. The use of both covariances and spectra are discussed.

THEORY

Let $R(t)$ be a random variable which represents the number of particles that enter a region in a time interval $(0, t)$, and assume that $R(t)$ is a Poisson process with parameter λ, so that

$$p_R(i) = P[R(t) = i] = \frac{e^{-\lambda t}(\lambda t)^i}{i!}, \quad i = 0, 1, 2 \ldots \quad (1)$$

Let $N(t)$ be a random variable which represents the number of particles in the region at time t (henceforth called the population), and let T_j be a non-negative continuous random variable which represents the residence time of the jth particle in the region. Assume that the T_j's are