Optimal Scheduling of Complex Batch Processes

By

HENRY X. SWANSON

A DISSERTATION PRESENTED TO THE GRADUATE COUNCIL OF
THE UNIVERSITY OF FLORIDA IN PARTIAL
FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA
1972

ACKNOWLEDGMENTS

The author wishes to express his appreciation for the assistance and support of:

Professor F. P. May, who served as chairman of his supervisory committee, first introduced the author to the problems of batch scheduling, and who was his principal research advisor;

Associate Professor A. W. Westerberg who introduced the author to systems design and optimization and served on his research proposal committee;

Professor H. E. Schweyer who served on the author's supervisory committee and who helped him define his problem;

Assistant Professor J. B. Wallace who served on his supervisory committee;

Professor R. D. Walker, Jr. who served on his supervisory committee;

The Department of Chemical Engineering for support and for the educational contributions of its faculty and staff;

The University Computing Center which provided support.

TABLE OF CONTENTS

	Page
ACKNOWLEDGMENTS	ii
LIST OF TABLES	v
LIST OF FIGURES	vi
LIST OF SYMBOLS	viii
ABSTRACT	xi
CHAPTERS:	
I. THE PROBLEM	1
II. INTEGER PROGRAMMING	8
Problem Formulation	8
History	12
III. INTEGER PROGRAMMING USING DUALITY AND THE REVISED SIMPLEX METHOD	16
IV. PURE SCHEDULING PROBLEMS	24
Balas' Enumeration	25
Branch-Bound Enumeration	38
A Solution to the Storage Problem	43
Optimization Strategies	44
V. ENUMERATION OF MIX/SCHEDULING PROBLEMS	48
VI. RESULTS, CONCLUSIONS, AND RECOMMENDATIONS	60
Solution Efficiency	60
Results	64
Conclusions and Recommendations	70
APPENDICIES	73
A. THE INVERSE MATRIX FOR PURE SCHEDULING PROBLEMS	74

TABLE OF CONTENTS (Continued)

	Page
B. PROGRAMS	86
BIBLIOGRAPHY	147
BIOGRAPHICAL SKETCH	149

LIST OF TABLES

Table		Page
1	Table of Data for Sample Problem.....	30
2	Optimal Schedule for Sample Problem.....	38
3	Batch Costs for Sample Batch Product Mix Problem..	54
4	Batch Characteristics for Large Batch Product Mix Problem.....	55
5	Chemical Inventories and Batch Sizes.....	56
6	Feasible Schedule Found for Batch Product Mix Problem.....	57
7	Computational Results Using Several Integer Programming Methods.....	61
8	Optimal Product Mix and Schedule for Sample Problem 2.....	65
9	Schedule for Two Time Periods.....	66
10	A Set of Scheduling Decisions.....	69
11	Best Schedule Found for Sample Problem 3.....	71

LIST OF FIGURES

Figure		Page
1	A Simple Batch Processing Problem.....	3
2	Flow Chart for First Two Linear Programs.....	26
3	Flow Chart for Last Four Linear Programs.....	27
4	Final Flow Chart for Sample Problem.....	28
5	Flow Sheet for Intermediate Inventory.....	29
6	Early Stages in Solution of Sample Problem.....	34
7	Flow Chart Showing First Four Decisions Made in Solving Scheduling Problem.....	35
8	Flow Chart Showing Backtracking After Sixth Linear Program.....	36
9	Continuation of Sample Problem to First Feasible Schedule.....	36
10	Flow Chart for First Step in Solving Sample Problem Using a Branch and Bound Enumeration Procedure.....	39
11	Flow Chart for Branch-Bound Procedure up to Fourth Partial Solution.....	40
12	Flow Chart Including Fifth Partial Solution.....	40
13	A Partial Solution with Three Remaining Either/Or Choices.....	46
14	A Partial Solution Where Three Either/Or Choices Can Be Made at Once.....	47
15	Second Batch Product Mix Problem.....	59
16	Three Possible Forms for Columns of B.....	78
17	Flow Sheet for Subroutine PIVOT.....	89
18	Flow Sheet for Subroutine LINEAR.....	92
19	Flow Chart for Subroutine DEMINF.....	98
20	Flow Sheet for Subroutine BACKTR.....	112

LIST OF FIGURES (Continued)

Figure		Page
21	Vector LF Before and After Control Point 1.....	118
22	Vector LF After Iteration Loop 104.....	119
23	LF After Iteration Loop 105.....	119
24	LF After Control Point 11.....	120
25	LF After Variable LL Constrained to Zero and Variable L Freed.....	120

LIST OF SYMBOLS

A	-	matrix which defines constraints to scheduling problems.
a_i	-	column i of matrix A'.
AA	-	a matrix used by computer programs to determine nonzero components of A'. A(1,J) gives the number of nonzero components of the first column of A'. For A(1,J) = I, A(Z,J) through A(I+1,J) give the location of the nonzero components of the Jth column of A'. For A(1,J) = I, A(I+2,J) through A(2I+1,J) give the corresponding nonzero components of the Jth row of A'.
B	-	the basis matrix of A'. (See Chapter III.)
BB	-	The basis matrix for the results of first dual linear program where all of the dual variables are constrained to zero.
BI	-	a matrix used by computer programs which is the inverse of B.
B1	-	the product of -CC times B.
B1F	-	B1 for the result of the first dual linear program.
c_{ijk}	-	cost of producing the kth batch of chemical i in production unit j.
c'_{ijk}	-	cost per hour of storing kth batch of chemical i in production unit j.
C_i	-	completion time of a particular job i.
CC	-	cost vector used by computer programs. CC(i) is the cost corresponding to dual variable i.
d	-	cost vector for the dual to scheduling problems.
e_i	-	the vector which has all zero components except for the ith component which is one.
IK	-	a matrix used by computer programs to locate information. If a variable I represents a scheduling decision then IK(1,I-NBM) locates the row of IMP which contains information about decision I. For a particular N2, IK(2,N2) locates the list IK(3,N2) through IK(IK(2,N2),N2). IK(3,N2) through IK(IK(2,N2,N2) are parameters used along with subroutines DEMIN or DEMINF to reduce the size of scheduling problems.

IMP - matrix which locates production decisions and scheduling decisions for the computer programs. For a particular I1, IMP(I1,1) will locate production decisions. Let I = IMP(I1,1). In this case variables IMP(I1,2) through IMP(I1,I) will correspond to decisions to produce some batches. Let J = IMP(I1,I+1). In this case variables IMP(I1,I+2) through IMP(I1,J) will correspond to decisions about scheduling the batches.

IX - vector which is used by computer program to locate freed variables. A dual variable I is constrained to zero if IX(I) = .FALSE. The decision to free a variable I on a particular branch of a decision tree (See Chapter IV.) is made by setting IX(I) = .TRUE.

IXB - let I be a dual variable representing a production decision. If the decision is implied by another decision then one need not free variable I. (See Chapter IV.) The computer programs indicate that a production decision is redundant by setting IX(I) = .FALSE. and IXB(I) = .TRUE.

If variable I represents a scheduling decision then the variable is constrained to zero on a branch of the decision tree by setting IXB(I) = .FALSE. IXO(I) = .FALSE. and IXB(I) = .TRUE. indicate that variable I is currently constrained to zero but may eventually be freed if the decision is made to produce the batch which is scheduled by variable I.

IXO - For production decisions IXO(I) = .FALSE. indicates variable I is constrained to zero on the current branch of a solution tree.

For scheduling decisions IXO(I) = .FALSE. means that a variable is temporarily constrained to zero but may be freed if the decisions are made to produce the batches scheduled and IXB(I) = .TRUE.

IX1 - a vector used by computer programs to determine which variables are in the basis. The variables IX1(1) through IX1(M) are in the basis.

IN - equivalent of n for computer programs.

LF - a vector which contains a list of numbers used by computer programs and described in Chapter IV. For a particular I, the absolute value of LF(I) is a variable which has been either freed or constrained to zero. If LF(I) is negative the program has eliminated the opposite decision from consideration. The first member of the list is LF(2) and the last member is LF(LF(1)).

m	-	the number of constraints to the dual of a scheduling problem.
M	-	equivalent of m for computer programs.
MIMP	-	number of rows of matrix IMP.
n	-	number of variables in the dual to a scheduling problem.
NBM	-	number of production decisions similar to Equations (41) through (44).
R	-	a reactor used in a sample program.
S	-	a separator used in a sample program.
t	-	the time that a particular batch is started.
v	-	a variable which can take a value of either 0 or 1.
w	-	the vector of dual variables to a scheduling problem.
x	-	vector of varisbles for a product mix or scheduling problem.
X	-	vector containing the value of the basis variables for the written programs.
z	-	objective functior.
z_i	-	objective function for an incumbent solution.
τ	-	the time period over which a scheduling problem is to be optimized.

Abstract of Dissertation Presented to the
Graduate Council of the University of Florida in Partial Fulfillment
of the Requirements for the Degree of Doctor of Philosophy

OPTIMAL SCHEDULING OF COMPLEX BATCH PROCESSES

By

Henry X. Swanson

June, 1972

Chairman: Professor F. P. May
Major Department: Chemical Engineering

The chemical industry performs a number of its operations as batch processes. This paper presents a method of scheduling batch processes, and the techniques developed are especially efficient for solving batch scheduling problems found in the chemical industry.

Two related problems are unique to batch processes. One has the problem of scheduling operations if more than one chemical can be processed in the same machine. Sequential batch operations usually require intermediate inventory. The beginning intermediate inventory over any period determines which schedules are feasible and the final intermediate inventory is a function of the schedule used. One has the problem of determining which schedules should be used and how much intermediate inventory should be maintained.

The author modeled batch scheduling problems using linear production and inventory costs and linear constraints. Other researchers have presented methods of solving such problems but the author's method proved more efficient for chemical batch scheduling problems.

The models of scheduling problems were written as a linear objective function and a large number of linear constraints. A subset of the constraints would apply for any feasible schedule. The dual problem was also linear with a large number of variables. Only a subset of these variables would take on nonzero values for any feasible solution. The solution algorithm used the revised simplex method and a novel method of identifying which variables were held to zero. This solution method allowed one scheduling decision to imply another which turned out to be an important feature for improving solution efficiency.

Sample scheduling problems were solved where the objective was to find how many batches should be produced and then to schedule them. The author could not find a case where such problems were attempted by other researchers.

The solutions to simple problems indicated that machine slack time is an inherent feature of batch scheduling problems. The solutions also indicated that intermediate inventory costs frequently cause one to produce a more varied product mix than if there were no inventory costs.

CHAPTER I

THE PROBLEM

It was the purpose of this work to develop a method to schedule the operation of fairly complex batch chemical processes. Actually a method was developed to optimize a fairly large class of problems. The solution procedure was developed to be most efficient for the chemical processing problems because of the structure of such problems.

Batch chemical processing units are designed so that various chemicals can be fed into them and after processing the resulting products are removed. We will look at the problem created when a chemical plant can produce a variety of products in the same batch production units. Typically there are a number of operating units in a chemical plant and the units can, in general, be used to produce a number of different products. The products from some units are used as feed to other units and one has to decide how each unit should be operated.

The oil industry has had some success in using linear programming to optimize continuous production rates when a variety of possible products can be produced. If one assumes that there is a fixed cost per pound of chemical made in a production unit one gets a linear objective function.

$$\text{minimize} \sum_{ij} c_{ij} x_{ij} \tag{1}$$

Here x_{ij} is the number of pounds of chemical i produced in

production unit j and c_{ij} is the fixed cost per pound of chemical produced. Profit is treated as a negative cost.

One also has material and energy balance constraints which are of the form

$$A\,x \leq b \quad . \tag{2}$$

Here we are using matrix notation where A is a matrix, x is the vector with components x_{ij} and b is a vector.

These constraints together with the linear objective function form a linear programming problem. This linear programming problem becomes an integer programming problem if the chemicals are to be produced in batches. Batch processes differ from continuous processes in at least two important aspects. Production occurs only as an integer number of batches and successive production units do not necessarily operate at the same average production (or use) rates.

Equations (1) and (2) with x_{ij} integer might not be too difficult an integer programming problem. This product mix problem might even be solved by linear programming especially if it has a lot more variables than constraints.[1] One might not be able to schedule the result of this product mix problem.

The scheduling problem can be seen by looking at a very simple batch processing problem. Suppose a chemical process uses two batch units, R and S, to produce a final product. Suppose that R produces 300 pounds of product and that it takes 500 pounds of product from R to feed S. Suppose further that it takes three hours to produce a batch in R and four hours to produce a batch in S.

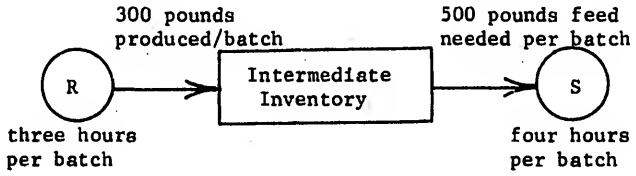


Figure 1. A Simple Batch Processing Problem.

With sufficient intermediate inventory, it is possible to produce 5 batches in R and 3 batches in S over a fifteen-hour period. If there is no intermediate inventory, the first batch of S cannot begin until after the second batch of R is completed. Thus, only two batches of S can be produced in a fifteen-hour period with no intermediate inventory.

One should notice that R is capable of producing 100 pounds per hour and S is capable of consuming 125 pounds per hour if there is sufficient feed. One should consider multiple uses for S since it is slightly oversized for the process shown in Figure 1.

Uriostie[2] considered the product mix problem given by Equations (1) and (2) together with x_{ij} integer. This product mix problem did not include scheduling constraints. Next Uriostie considered scheduling the problem in a second step. If one adds restrictions on the number of batches produced to Equation (2) eventually one will get a product mix that is schedulable.

Thus far we have not considered any inventory costs in our problem formulation. In general we will be concerned with the problem where the cost of each product in inventory is proportional to the amount of product and to the time that this product is in inventory.

The real product mix problem for batches should include Equation (1) and a cost for inventory. The constraints should include Equation (2) and scheduling constraints. An efficient method will be presented for solving this batch product mix problem. Actually the solution procedure turns out to be useful for solving a larger class of problems than the batch operation problems which are considered in detail in this thesis.

We want a solution procedure that will be very efficient for solving certain problems found in the chemical industry. Three different types of problems are considered in this work.

The petroleum industry uses the product mix problem in order to set yearly production goals and sales goals for continuous processes. The author would like to be able to solve the batch product mix problem for the same reason. Sales and production goals are useful and the batch product mix problem can provide this information.

The author would like to be able to solve the batch product mix problem and two other related problems. The other two problems are caused when sales goals are not met exactly. The production department could use the result of the yearly batch product mix problem if sales goals were met exactly.

Actually sales vary from month to month and sales departments are encouraged to exceed the sales' goals for the most profitable products. The production department has to be able to fill whatever sales commitments are made each month.

The production department can use a modified batch product mix problem for monthly operations. This problem would again use

costs for making each batch and inventory costs. The batch product mix problem would have the same form as the yearly batch product mix problem except for some additional constraints. These constraints would force production to fill the existing sales orders.

The last problem the author would like to be able to solve is simply a scheduling problem. The pure scheduling problem is formed when one decides what operations should be performed and one merely needs to schedule the operations. This is the type of problem Uriostie[2] solved for the case of two machines and two products. Uriostie also assumed he had sufficient intermediate inventory to begin any operation.

In reality the inventory left over from one month's production affects production in the next month. We will only be interested in optimizing operations over a fixed time period and will ignore the effect of these operations on some sequential time period.

Our scheduling problem is related to another problem called in the literature "the job-shop problem."[3] The job-shop problem comes from the problem of scheduling production in a machine shop where pieces of metal have to be worked on different machines. The problems are different however in that in our problem there is in general no unit of chemical that has to be worked on. Three-sevenths of one batch might be needed to make a batch of feed for the next processing unit, for example, and frequently a batch separation process will produce some odd fraction of material to be recycled.

One may have to do a number of operations on a piece of metal before the job of working the metal is completed. Other authors

reported work on scheduling problems but interest has most often centered on a regular measure of performance for the job-shop problem. A regular measure of performance can be expressed as a function of the job completion times.

$$M = f(C_1, C_2, \ldots, C_n) \tag{3}$$

C_i is the time that job i is completed and a regular measure of performance also has the characteristic that the measure increases only if at least one of the completion times increases. That is, if $M' = f(C_1', C_2', \ldots, C_n')$, then $M' > M$ only if $C_i' > C_i$ for at least one i.

Conway, Maxwell, and Miller[3] suggest one of the real reasons that so much interest has centered around minimizing the completion time of the last job (a regular measure of performance) is that a simple algorithm was developed for minimizing this completion time in a two-machine job-shop. Researchers, however, have not found a simple algorithm for more complex cases. We will be using an inventory cost which is proportional to the amount of each material in inventory and the time it is stored. This is not a regular measure of performance since the cost is minimized if we start jobs creating inventory as late as possible. One reason why this inventory cost makes sense is that if a product cannot be shipped and sold until a certain due date, there is no reason why the company should spend its resources completing the job early.

Some heuristic procedures have been developed for solving the job-shop problem. Actually heuristic procedures are a fairly good idea for scheduling problems.[4] The job-shop scheduling problem can be

formulated as a 0-1 programming problem but as late as 1963 realistic problems could not be solved.[5] Lasdon[6] has some encouraging results for large scale problems but his scheduling problems are not as difficult as the ones this work will attempt to solve.

In one way, our scheduling problem is harder than the job-shop scheduling problem; we do not have a fundamental unit of the chemical product which has to go through a certain sequence of machine operations. Our scheduling problem is easier because usually one has a number of batches of the same type and this reduces the dimensionality of the problem. (Knowing that batch 1 of type E proceeds batch 1 of type F implies that batch 1 of type E proceeds all of the batches of F and reduces the size of the problem.)

Multiple batches (that is, continuing use of the process) are the rule in the chemical industry because of the high development costs of a chemical process. The high costs involved justify a fairly elaborate effort to find a good product mix and a good schedule.

Graham[7] also looked at a similar scheduling problem. No inventory costs were assessed but costs were included for setting up each type of batch. A modification of dynamic programming was used. There are computer storage problems for dynamic programming problems of any complexity and Graham circumvented some of them by an approximation technique. The author questions whether the storage problems can be avoided for more complex problems than Graham attempted. The author's enumeration scheme also looks more promising than Graham's because of the simplifications caused by multiple batches.

CHAPTER II

INTEGER PROGRAMMING

A solution procedure is needed for product mix problems with or without sales commitments and for pure scheduling problems. An enumeration procedure similar to integer programming is developed in this work. Some of the nomenclature of integer programming will be useful for explaining the new enumeration procedure. First our problems will be formulated as integer programming problems and then some of the history and terminology of integer programming will be discussed.

Problem Formulation

The cost of producing batches is expressed in Equation (1) for a product mix problem. For a pure scheduling problem the number of batches of each type to be produced is fixed so this summation is a constant. We can express the number of batches produces as a sum of 0-1 variables.

$$x_{ij} = \sum_{k} x_{ijk} \tag{4}$$

where x_{ijk} = 1 if the kth batch of i in j is to be produced

= 0 otherwise

0-1 variables will simplify later discussion.

We will need to include inventory costs in our inveatigations. Let us look at the contribution to the inventory cost over a fixed period, τ, caused by one batch. Let t_{ijk} be the time that batch k of i is started in unit j. Suppose it costs c_{ijk} per unit time to store

the feed for the batch in inventory. If we do not start the batch until τ the cost for storing the feed is $c_{ijk}\tau$. The cost for feed inventory will decrease by $c_{ijk}(\tau - t_{ijk})$ as a consequence of starting the batch at t_{ijk}. Suppose that it costs c'_{ijk} per unit time to store the product from the batch in inventory and the batch has a processing time of p_{ijk} in the unit. The overall affect upon inventory cost due to starting the batch at t_{ijk} is

$$-c'_{ijk}(t_{ijk} + p_{ijk} - \tau) + c_{ijk}(t_{ijk} - \tau) \quad . \tag{5}$$

The inventory costs given by Equation (5) summed over all i, j, and k is a linear objective function which we need in order to have an integer program. The total objective function for our problem is the sum over i, j, and k of Equation (5) and the costs given by Equation (1). Next we need to formulate our constraints.

Material Balance

A material balance gives a simple linear constraint between batches.

$$\sum_{ijk \epsilon P_I} w_{ijk} x_{ijk} \geq \sum_{ijk \epsilon C_I} w_{ijk} x_{ijk} \tag{6}$$

where w_{ijk} is the amount of chemical produced or used by batch k of i in j

P_I is the set of batches producing a kind of chemical

C_I is the set of batches consuming the same kind of chemical.

Completion Time

We are to optimize operations over a fixed time period. A simple constraint will make sure that there is enough time to produce the batches in each production unit.

$$\sum_{ik} p_{ijk} x_{ijk} \leq \tau \tag{7}$$

We need to assure that batches are completed by the due date if sales commitments are made. For the pure scheduling problem every product batch has a due date.

$$t_{ijk} + p_{ijk} \leq d_{ijk} \tag{8}$$

where d_{ijk} is the due date for producing batch k of i in j.

We need to use some tricks with our problem formulation to make sure that no inventory cost is assessed if we make the decision not to make a batch. We can use a synthetic device that will schedule any batch which will not be made ($x_{ijk} = 0$) to start at the final time τ. We do not want a contribution to the inventory cost if $x_{ijk} = 0$. This can be done by the constraints:

$$t_{ijk} \leq \tau \tag{9}$$

$$t_{ijk} + \tau x_{ijk} \geq \tau \tag{10}$$

and adding the constant $c'_{ijk} p_{ijk}$ to the objective function.

When $x_{ijk} = 0$, Equations (9) and (10) force $t_{ijk} = \tau$. One can look at Equation (5) to see that the contribution to the inventory cost when $t_{ijk} = \tau$ is 0.

Equation (10) is nonrestrictive when $x_{ijk} = 1$.

Scheduling

This work is primarily concerned with scheduling different operations on the same piece of equipment. The equipment can only operate on one thing at a time so one has the constraints;

$$\text{either/or} \quad \begin{matrix} t_{ajb} + p_{ajb} \leq t_{cjd} \\ t_{cjd} + p_{cjd} \leq t_{ajb} \end{matrix} \quad . \tag{11}$$

Equations (11) are concerned with two batches to be made in the same piece of equipment. Equations (11) say that one of the batches has to be finished before the other begins. This either/or choice can be expressed using a 0-1 variable.

$$\begin{aligned} & t_{ajb} + p_{ajb} \leq t_{cjd} + (t_{ajb} + p_{ajb})v \\ & t_{cjd} + p_{cjd} \leq t_{ajb} + (t_{cjd} + p_{cjd})(1 - v) \\ & v = 0 \text{ or } 1. \end{aligned} \tag{12}$$

To form an integer program we also would have to put some constraints on our variables like v in Equations (12). For example, we would need constraints to force a batch of A before all the batches of B if the decision is made to schedule A before the first batch of B. Equations (12) only hold if one has decided to produce batch b of a in j and batch d of c in j. One may or may not decide to produce

a batch in the product mix problem, therefore Equations (12) are not correct for the product mix problem. A more complex set of equations has to be written for the product mix problem.

In general it is difficult to correctly formulate integer programming problems. One of the most frequently encountered discrete problem, the traveling salesman problem,[8] was recognized and solved years before someone saw how to formulate it as an integer programming problem. The actual enumeration procedure developed later avoids much of the difficulty of problem formulation.

History

A great deal of work has been done on integer programming. Reviews of the literature[4,8] have been done by others and a general review will not be attempted here. Rather we will look at some of the major types of solution procedure and their limitations. As the new enumeration procedure is developed in a later chapter the relationship will be shown between the new method and some of the more successful methods of integer programming.

One of the earlier integer programming methods was developed for the all-integer problem by Gomery.[9,10] In this method all the non-integer constraints are written. This problem is solved by a linear programming method and new constraints are added to the problem. These new constraints reduce the constraint set to a smaller set containing the valid integer solutions. Constraints are added until an all-integer solution is found. A great deal of work was done refining this cutting plane method but unfortunately the method never proved to be very useful

because of the large number of constraints which had to be added to the problem.

One of the most recent integer programming methods is also due to Gomery.[11] Again in this method an integer programming problem is written without the integer constraints and one wants to reduce this convex set to find the integer solutions. Group theory is used and some refinements have been made in the method but it also doesn't work very well because of computer storage problems.

Most, if not all, of the currently more successful algorithms for solving integer programming problems are implicit enumeration schemes.[12] Land and Doig[13] were the first to look at this method. For implicit enumeration one first solves the problem using linear programming without the integer constraints. One then looks at the effect of adding integer constraints to the problem. One might, for example, look at adding the constraint $x_7 = 4$ to the problem where x_7 must be an integer variable. If $x_7 = 4$ makes the problem infeasible one need not look at any more integer solutions with $x_7 = 4$. If $x_7 = 4$ gives a feasible solution one might solve the linear programming problem with $x_7 = 4$ and the original constraints and then look at adding additional integer constraints to this result.

An additive algorithm due to Balas[14] is similar to one of the enumeration schemes in a later chapter. Suppose one has a problem with 0-1 variables. Equation (4) shows how to change a problem with integer variables into one with 0-1 variables. One enumeration scheme starts by first solving the linear programming problem without integer constraints. If the solution is all integer then you are finished. If

not the scheme adds integer constraints one at a time. One chooses a first variable, v_1, and constrains it to be zero. The problem is solved with all noninteger constraints and $v_1 = 0$. If the problem is infeasible then one need never look at solutions with $v_1 = 0$ and one need only look at $v_1 = 1$. If the problem has a solution with $v_1 = 0$ we call this a partial solution. We next need to find if there are any feasible solutions to the original problem with $v_1 = 0$. We do this by adding more integer constraints to our partial solution. Suppose we add the consecutive constraints $v_2 = 0$, $v_3 = 0$, ..., $v_n = 0$ and get an infeasible solution. We say that there are no feasible completions to the partial solution $v_1 = 0$, $v_2 = 0,\ldots,v_n = 0$. We next backtrack and look for feasible completions to the problem $v_1 = 0$, $v_2 = 0,\ldots,$ $v_{n-1} = 0$, $v_n = 1$.

The process of adding constraints to the problem until one finds a solution or proves there is no feasible completion is called fathoming.[14] Once one finds a solution to the original problem the fathoming and backtracking procedure can be made more efficient. We can use the objective function, z^*, of the solution to the original problem. We need not look for any solutions which do not have objective functions better than z^* since we are trying to find the best possible solution. We only keep the objective function from the best solution we have found so far. If we find a new solution with a better objective function we use it for the new z^*. Suppose at some stage of our fathoming procedure we reach a partial solution which has an objective function which is not as good as z^*. At this point we can backtrack because adding constraints will not improve the objective function of

the partial solution and we are only looking for feasible completions with objective functions better than z^*.

In order to implement an efficient enumeration scheme to solve an integer programming problem, one needs information on the effect of adding a constraint to a linear programming problem. An integer programming method is developed in Chapter III which gives a great deal of information on the effect of adding constraints to a problem. In Chapters IV and V the integer programming method will be shown to be very efficient for batch scheduling problems.

CHAPTER III

INTEGER PROGRAMMING USING DUALITY AND THE REVISED SIMPLEX METHOD

A new method has been developed to solve integer programming problems. The method is especially efficient for solving scheduling problems. In Chapter II scheduling problems were shown to have two general types of constraints. The first type of constraint includes the material balance constraints and the completion time constraints. Any feasible schedule has to satisfy all of the material balance constraints and all of the completion time constraints. A feasible schedule only satisfies one of the scheduling constraints from each pair of scheduling constraints of the second kind. The new solution method solves the dual of the scheduling problem. The dual problem contains paired variables corresponding to the paired scheduling constraints in the primal problem. One variable from each pair of variables is constrained to zero. The new solution method first solves the dual problem without any paired variables and later successively adds variables to the problem. The dual problems are solved by means of the revised simplex method[15] which greatly reduces the amount of work required for scheduling problems.

The new integer programming method will be explained by means of a sample scheduling problem. Suppose we have to schedule three operations, E, F, and G, on a particular machine. Suppose that E produces inventory that costs 30 units of value for each unit of time stored in inventory, F produces inventory that costs 40 units for

each unit of time in inventory, and G consumes inventory that costs 30 units of value for every unit of time in inventory. The objective function will be of the form

$$\text{minimize} \quad -30t_E - 40t_F + 30t_G + K \quad . \tag{13}$$

t_E, t_F and t_G are the times that batches E, F, and G are started. As shown in Equation (5) K will be a constant if we are to optimize our scheduling over a fixed time period.

We will suppose G has to start before 60, F has to start before 40 and E has to start before G. Suppose further that E and F take 20 and G takes 10 units of time. We have three fixed constraints.

$$t_G \leq 60 \tag{14}$$

$$t_F \leq 40 \tag{15}$$

$$t_E + 10 \leq t_G \quad . \tag{16}$$

We also need to have scheduling constraints that do not allow two batches to be produced concurrently in the same machine.

$$\text{either/or} \quad \begin{matrix} t_E + 20 \leq t_F \\ t_F + 20 \leq t_E \end{matrix} \tag{17}$$

$$\text{either/or} \quad \begin{matrix} t_F + 20 \leq t_G \\ t_G + 10 \leq t_F \end{matrix} \quad . \tag{18}$$

The scheduling problem is now formulated as a set of linear programs. The programs can be written in a compact form.

$$\text{minimize} \quad K - (30, 40, -30)t$$

$$\text{such that} \quad \begin{Bmatrix} 0 & 0 & -1 \\ 0 & -1 & 0 \\ -1 & 0 & 1 \\ -1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & -1 \end{Bmatrix} t \geq \begin{Bmatrix} -60 \\ -40 \\ 20 \\ 20 \\ 20 \\ 20 \\ 10 \end{Bmatrix} \qquad t \geq 0 \qquad (19)$$

We have four possible linear programs, depending on which constraints are included from the last two pairs of constraints. Corresponding to each of these linear programs there is a dual linear program.[15]

	minimize $c't$		maximize $d'w$	
primal problem	$A \; t \geq d$ $t \geq 0$	dual problem	$A' \; w \leq c$ $w \geq 0$	. (20)

The objective function of the solution to a dual problem has the same value as the objective function of the solution to the corresponding primal problem. In fact, the same information can be retrieved from a solution to the dual problem as can be retrieved from a solution to the primal problem.

The dual problems corresponding to (19) can also be written in a compact form.

$$\text{maximize} \quad (-60, -40, 20, 20, 20, 20, 10)w + K$$

$$\text{such that} \quad \begin{matrix} 0 & 0 & -1 & -1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & -1 & 1 \\ -1 & 0 & 1 & 0 & 0 & 1 & -1 \end{matrix} \quad w \leq \begin{matrix} -30 \\ -40 \\ +30 \end{matrix} \quad w \geq 0 \qquad (21)$$

If a constraint is used in a primal problem the corresponding variable will be used in the dual. If a constraint is not used in the primal then the corresponding variable will not be found in the dual.

In order to use linear programming we first rewrite Equation (21) in terms of equality constraints.

$$\text{maximize} \quad (-60, -40, 20, 20, 20, 20, 10, 0, 0, 0)(w_1 s) + K$$

$$\begin{matrix} 0 & 0 & -1 & -1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & -1 & 1 & 0 & 1 & 0 \\ -1 & 0 & 1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 \end{matrix} \begin{matrix} w \\ s \end{matrix} = \begin{matrix} -30 \\ -40 \\ +30 \end{matrix} \qquad (22)$$

$$\begin{matrix} w \\ s \end{matrix} \geq 0$$

The first step of the solution procedure will be to solve (22) without any of the either/or variables. There are a number of algorithms for solving linear programming problems but the revised simplex method is especially useful for our needs.

Linear programming algorithms use a stepwise method of finding an optimal solution. Primal simplex methods first find an extreme point of the convex constraint set. One then pivots between adjacent extreme points to the optimal solution. If a linear programming problem has n variables and m constraints, the extreme points are characterized by n-m of the variables having a value of 0. The m remaining variables are called basis variables and one moves from one extreme point to

another by changing the variables in the basis. One needs to know how much the objective function will change if a new variable is put in the basis. One uses a vector y_j for each variable, j, in order to determine the change in the objective function. Suppose we are doing the dual problem of Equations (20). For each variable n_1 let a_{n_1} be the corresponding nth column of A'. Let b be the set of basis variables. y_j expresses the jth column of A' in terms of the basis columns of A'

$$a_j = \sum_{i \in b} y_{ij} a_i \tag{23}$$

Equation (22) can also be expressed in matrix notation

$$a_j = B\, y_j \tag{24}$$

Here B is the matrix formed from the basis columns of A'.

In the normal simplex method a new y_j is generated for each j for each step of the solution process. The revised simplex method generates an inverse matrix, B^{-1}, with each step of the solution process. The vectors, y_j, can be generated by multiplying B^{-1} times the columns of the original constraint set.

Now we are going to first solve problem (22) without any of the either/or variables. While we are solving the problem without the either/or variables we do not need the information, y_j, about them. If the revised simplex method is used, the information, y_j, does not have to be generated during the first part of the solution process.

The enumeration process will be carried out in the dual. At

first the problem will be solved without either/or variables and later some of these variables will be added to the problem. The necessary information for adding vectors to the problem can be generated when needed by the revised simplex method. Other linear programming methods need to generate the information at every step of the solution process which is a lot of unnecessary work.

It should be noted that there is no need for integer variables. Greenberg[16] also noted that you don't need integer variables with either/or constraints like Equations (17) and (18). However, Greenberg did not solve his problem in the dual and had to solve a new linear program every time he added a constraint.

The solution to Equation (22) without the either/or variables has an objective function of K-1000. Information from the revised simplex method shows that two of the either/or variables can be added without increasing the objective function. These two variables correspond to the first of Equations (17) and the second of Equations (18). From this solution of Equation (22) without the either/or constraints we can find the primal solution ($t_A = 0$, $t_B = 40$, $t_C = 20$). The either/or constraints scheduling A before B and C before B are satisfied by this solution.

We will develop several enumeration procedures in the next chapter. In all of the procedures we will use the dual and the revised simplex method. Variables corresponding to either/or constraints will be added to existing solutions to linear programs in the dual. In order to make decisions, one needs information on the effect of adding new variables to an existing solution. It takes work to generate information and one has to decide how much information is needed.

There are three types of information available on the effect of adding a variable to a problem. One can solve the linear program with the new variable. This takes some work and we will need information on a large number of variables. Most integer programming algorithms use much less information. In the simplex method one improves a solution by moving from one extreme point of the feasible solutions to another. One moves along an "edge" of the region of feasible solutions from one extreme point to another. The simplex method will move along the "edge" that has the largest feasible gradient to improve the objective function. Most integer methods use this gradient to make decisions. In the revised simplex method one has to multiply two vectors together in order to determine the gradient. If the problem has m constraints then the vectors have m+1 components. One can also determine how much the objective function will increase as a result of moving from one extreme point to another. In the revised simplex method, one has to multiply a matrix times a vector in order to determine the cost of moving from one extreme point to another. We will use this information.

We will be multiplying a m+1 dimensional square matrix times a vector and this normally involves $(m+1)^2$ multiplications and the same number of additions. For scheduling problems most of our vectors will come from constraints like Equations (17) and (18) which means they will have only two nonzero terms, plus and minus one. Our multiplication will consist only of subtracting the components of one row of the matrix from another. This simplification decreases the difficulty of determining the cost of moving to a new extreme point.

There is another reason for determining the cost of moving to a new extreme point when a vector is added to a problem. Suppose we are to solve problem (19) with a different objective function. We might solve the problem without any of the either/or constraints and then consider adding the constraint that forces F to be completed before E. Since E has to start before G, adding this constraint also forces F to be completed before G. We are essentially making two decisions at once if we decide to force F to be completed before E.

For the enumeration procedures in the next chapter one decision will frequently have multiple consequences. The importance of these decisions justifies additional work in addition to simply calculating gradients of the objective function. This will be shown in later chapters.

CHAPTER IV

PURE SCHEDULING PROBLEMS

In Chapter III a method was developed to solve integer programming problems that was particularly well suited to solving scheduling problems and the method was illustrated by means of a sample scheduling problem.

In this chapter we will be concerned only with scheduling batch processes. We will not be concerned with how many batches are to be made. We will assume we know how many batches are to be made in each unit over a fixed time period and will try to find the best schedule. This problem will be called the pure scheduling problem.

In Chapter III we solved a pure scheduling problem by adding constraints one at a time to linear programming problems. At this point we want to find the best of all possible schedules. One has to find this optimal schedule and prove that it is optimal. One does this by proving that no other schedule has an objective function better than the objective function of the optimal schedule.

We also want to be able to find the best schedule and prove its optimality in as few steps as possible. The amount of work involved is a function of the order in which constraints are added to our linear programming problems. We will look at two general methods of adding constraints. One enumeration method developed by Balas[14] may not be as efficient as the other method but does not require much storage of information when implemented on a computer. The other method, the branch-bound method, was essentially developed

by Land and Doig.[13] The branch-bound method requires more storage of information than the Balas enumeration method but it usually requires enumeration of fewer schedules. We will first look at the Balas enumeration method.

Balas' Enumeration

In solving a scheduling problem constraints are applied one at a time. We want to find all the possible schedules or find some of the schedules and then prove that none of the remaining schedules are better than one of the schedules we have already found. A systematic way of investigating or eliminating all the possible schedules is needed.

An example will be useful to show the Balas enumeration method of finding the optimal schedule. We will use the example that was developed earlier.

$$\text{minimize } -30E - 40F + 30G$$

$$\text{either/or} \qquad E + 20 \leq F \tag{25}$$

$$F + 20 \leq E \tag{26}$$

$$\text{either/or} \qquad F + 20 \leq G \tag{27}$$

$$G + 10 \leq F \tag{28}$$

such that

$$G \leq 60 \tag{29}$$

$$F \leq 40 \tag{30}$$

$$E + 20 \leq G \tag{31}$$

Here we have a problem with seven constraints. The feasible schedules will satisfy either Equation (25) or Equation (26). They also have to satisfy Equation (27) or Equation (28). Suppose we solve the problem with none of the first four constraints applied. Next we might solve the linear programming problem with the first constraint added. Finally, we might find the schedule is optimal when Equation (27) is also added. In our example for these first three linear programming problems, the objective function is -1000. One method of drawing a flow chart for these three steps is given below.

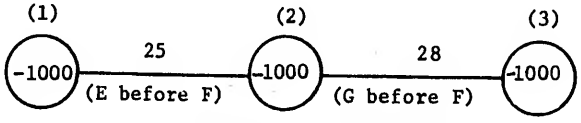


Figure 2. Flow Chart for First Two Linear Programs.

Using a computer it is necessary to store the same information given in the above graph. A computer could store the list of numbers (26, 25) when it started the second linear program. This would signify that Equation (25) was the only either/or constraint in effect during the program. Then the machine could store the list of numbers (26, 25, 27, 28) when it started working on the third linear program. One would want to store the results of the first feasible schedule obtained.

The third linear program gives the best possible schedule that satisfies Equations (25) and (28). We need to find the best of all

the posaible schedules with Equation (25) satisfied that also solve the original problem. In the language of integer programming we say that we need to fathom all the solutions obtained from the partial solution satisfying Equation (25).

When we start the linear program to solve the linear programming problem satisfying Equations (25) and (27) the corresponding flow chart might be drawn as below.

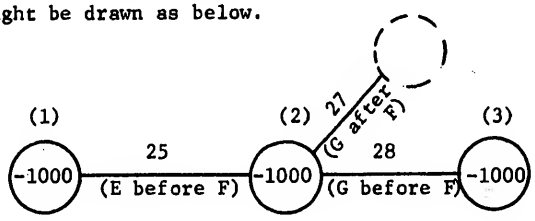


Figure 3. Flow Chart for Last Four Linear Programs.

We could store the same information in the computer in the form of the list of numbers (26, 25, -27, -28). The negative sign indicates Equations (27) and (28) have both been tried. We only did enough work on the solution satisfying Equations (25) and (27) to prove that the solution cannot have a value for the objective function smaller than for the objective function for the feasible schedule we have already obtained. Actually the best schedule satisfying Equations (25) and (27) has an objective function of -400, but since the optimization procedure did not find this schedule Figure 3 does not have -400 in the fourth circle and the circle is not completed.

After we have fathomed all solutions from the partial solution satisfying Equation (25) we need to look at solutions satisfying Equation (26). The phrase used in integer programming is that we back-track and find the partial solution satisfying Equation (26). Actually

we only need to prove that no solution satisfying Equation (26) is better than the solution we have already found. When we backtrack we would store the list (-26, -25) in the computer since the only information we need is that we have fathomed the solutions satisfying Equation (25) and we are working on fathoming the solutions satisfying Equation (26). The best schedule satisfying Equation (26) has an objective function of -200. The final flow chart is given below and the computer would no longer have any numbers in the list it was storing.

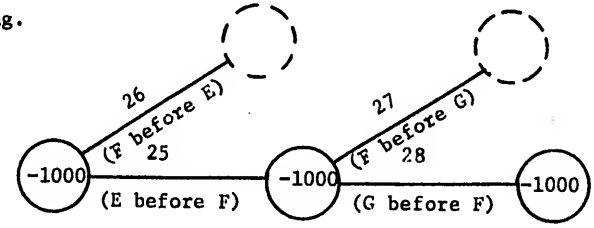


Figure 4. Final Flow Chart for Sample Problem.

Other pure scheduling problems can be solved in a similar manner. One needs only to apply a constraint from each of the either/or pairs and then solve the linear programming problem. One then backtracks and fathoms until all schedules are either found or eliminated. This Balas enumeration method of solving the problem has the advantage that we never have to store more than one solution and one list. Other advantages and disadvantages will be discussed when we look at another method of solving the problem. The Balas enumeration method used here fathoms partial solutions until the entire problem is solved. The other method (branch-bound) of solving the problem may run into

storage problems and then we can use the method developed here to fathom partial solutions found by the other method.

A program has been developed and implemented in the Fortran language to solve problems in the manner illustrated by the previous sample problem. The linear programs are solved in the dual in a manner similar to that given in Chapter III. To show some of the details of how the program works we will again look at an example.

We will suppose that two products, E and F, can each be made in two reactors, R1 and R2, and that the product from each reactor is held in intermediate inventory until it is processed by a separator. For the purposes of the problem we will suppose that output from the separator can be sold immediately so there is no inventory created by the separator and in this problem that we do not have to keep feed for the reactors in inventory.

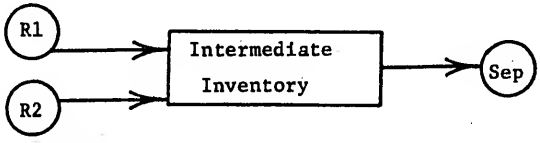


Figure 5. Flow Sheet for Intermediate Inventory.

We will be concerned with batch units and suppose that we know the time it takes to produce each batch and the amount it costs to store a batch of feed for the reactors or the separator.

Table 1

Table of Data for Sample Problem

Batch	ER1	ER2	ES	FR1	FR2	FS
Time to produce batch	3.7	4.2	2.1	4.6	4.7	2.6
Cost of keeping batch one unit of time in inventory	5.0	5.1	5.0	6.2	6.4	6.0

Let's assume that we produce a batch of E in each reactor over a period of time and a batch of F in each reactor over the same period. Also assume that we are to process two batches of E and two batches of F in the separator. Let the variables be:

t_{ER1} - the time we start the batch of E in reactor 1

t_{ER2} - the time we start the batch of E in reactor 2

t_{E1S} - the time we start the first batch of E in the separator

t_{FR1} - the time we start the batch of F in reactor 1

etc.

We will have the constraints that the first batch of either kind through the separator will precede the second batch of either kind.

There are the constraints that two batches cannot be processed through the separator at the same time. For instance, either the first batch of E through the separator has to be completed before the first batch of F begins or the first batch of F has to be completed before the first batch of E begins.

$$\text{either/or} \quad \begin{matrix} t_{E1S} + 2.1 \leq t_{F1S} \\ t_{F1S} + 2.6 \leq t_{E1S} \end{matrix} \tag{32}$$

There are six pairs of either/or constraints of this type.

We will suppose that all of the batches are completed by the time 9.5. This gives the constraints:

$$\begin{matrix} t_{ER1} + 3.7 \leq 9.5 \\ t_{ER2} + 4.2 \leq 9.5 \\ t_{ES1} + 2.1 \leq 9.5 \\ \text{etc.} \end{matrix} \tag{33}$$

Assume also that there is not enough inventory to feed the second batch of either chemical through the separator until we have finished making some of the chemical in a reactor.

$$\begin{matrix} \text{either/or} & \begin{matrix} t_{ER1} + 3.7 \leq t_{E2S} \\ t_{ER2} + 4.2 \leq t_{E2S} \end{matrix} \\ \text{either/or} & \begin{matrix} t_{FR1} + 4.6 \leq t_{F2S} \\ t_{FR2} + 4.7 \leq t_{F2S} \end{matrix} \end{matrix} \tag{34}$$

Of course the first batch of E has to be completed in the separator before the second batch begins. The first batch of F in the separator has to be completed before the second batch ends.

$$\begin{matrix} t_{S1E} + 2.1 \leq t_{S2E} \\ t_{S1F} + 2.6 \leq t_{S2F} \end{matrix} \tag{35}$$

Our problem is a minimization problem minimizing

$(-5.0t_{ER1} -5.1t_{ER2} +5.0t_{E1S} +5.0t_{E2S} -6.2t_{FR1} -6.4t_{FR2} +6.0t_{E1S} +6.0t_{E2S})$

subject to constraints like Equations (33) and (35). In addition the optimal schedule satisfies one constraint from each pair of constraints from Equations (32) and (34).

As was done earlier our problem is solved in the dual. The dual problem has some variables associated as pairs.

Primal

minimize $c \cdot t$ such that

$$\text{either/or constraints} \quad A_1 t \leq \underline{b}_1$$
$$A_2 t \leq \underline{b}_2 \qquad (36)$$
$$t \geq \underline{0}$$

Pairs of constraints from Equations (32) and (34) make up the set $A_1 t \leq \underline{b}_1$. Only one constraint from each pair has to be satisfied.

Dual

minimize { maximize $b_1 x_1 + b_2 x_2$ such that

$$A_1' x_1 + A_2' x_2 \geq \underline{c} \qquad (36)$$
$$x_1 \geq \underline{0} \quad x_2 \geq \underline{0}$$

and one member of each set of paired variables if x_1 has to be constrained to 0.

We minimize by finding the set of variables of x_1 which can be constrained to 0 and the maximization problem will have a minimum value.

In the dual the pairs of associated variables correspond to the pairs of associated constraints from Equations (32) and (34). The final solution in the dual will have one member of each pair of associated variables free to take on any nonnegative value and one member of the pair constrained to zero.

First, the problem is solved with all the pairs of associated variables constrained to zero. This corresponds to solving the primal problem without constraints from Equations (32) and (34). The linear programming problem resulted in an objective function of -91.03.

The result of a linear programming problem with n variables and m constraints will be that n - m of the variables will be zero and, except for a trivial case, some of the m basis variables will be nonzero and positive since required by the problem. Linear programming provides a rather simple method of removing one variable from the set of basis variables and finding a feasible solution to the problem with a different variable in the basis. The program uses this method to find how much the objective function will change if one variable is removed from the basis and one of the either/or variables is put in the basis. For example the variable corresponding to the constraint,

$$t_{ER1} + 3.7 \leq t_{S2E} \quad , \tag{37}$$

can be put in the basis and the objective function will increase to -54.03. Actually the dual problem (Equation (36)) was formed and Equation (37) was represented by variable one in the dual. We can find how much the objective function will increase by putting any one

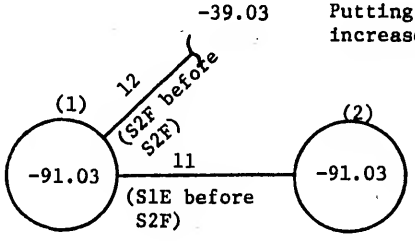
member of the either/or pairs in the basis. The program found that putting variable 12 in the basis would increase the objective function more than putting any other member of the either/or pairs in the basis. As the problem was formed variable 12 corresponded to the constraint;

$$t_{S2F} + 2.6 \leq t_{S1F} \quad . \tag{38}$$

The other member of the either/or pair, variable 11, corresponded of course to the constraint;

$$t_{S1E} + 2.1 \leq t_{S2E} \quad . \tag{39}$$

The objective function increased to -39.03 when variable 12 was put in the basis so there was a good chance that a solution to the problem would be found with a smaller value of the objective function. The linear program was solved with variable 11 free and variable 12 constrained to be zero.



Putting variable 12 in basis will increase objective function to -39.03.

Optimal schedule with S1E before S2F. Result of linear program gave objective function of -91.03 with variable 11 free and variable 12 constrained to zero.

Figure 6. Early Stages in Solution of Sample Problem.

The linear program with all either/or variables except 11 constrained to zero gave an objective function of -91.03. We repeat ourselves by finding how much the objective function will increase by adding one more member of the either/or set to the basis. We again find the variable, x_{13}, which will increase the objective function the most if it is added to the basis. We find that x_{13} and x_{14} make up an either/or pair and we allow variable x_{14} to be free while constraining x_{13} to be zero.

The computer initially stored the numbers (12, 11) in a list to record that solutions with variable 11 free were to be fathomed and that solutions with variable 12 free were to be fathomed next. A flow chart of the first several decisions is shown and the list is given.

Computer list (12, 11, 13, 14, 3, 4, 7, 8)

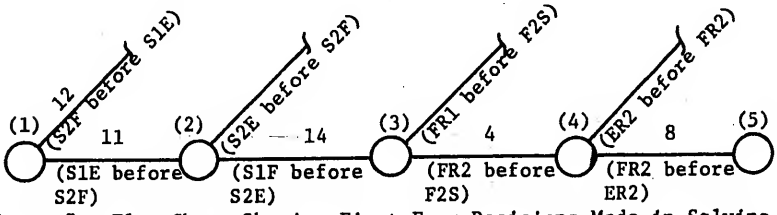


Figure 7. Flow Chart Showing First Four Decisions Made in Solving Scheduling Problem.

On the next step we find that variable x_1 will increase the objective function the most if it is added to the basis. We constrain variable x_1 to be zero and solve the linear programming problem with variable x_2 free. This linear programming problem turns out to have an unbounded solution so we backtrack and look at solutions with x_1 zero and x_2 free. An unbounded solution in the dual corresponds to infeasibility in the primal.

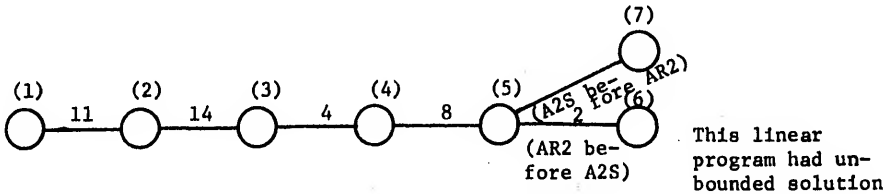


Figure 8. Flow Chart Showing Backtracking After Sixth Linear Program.

We continue our solution procedure until we arrive at the first feasible solution and then we start to backtrack. The first feasible solution had an objective function of 10.65.

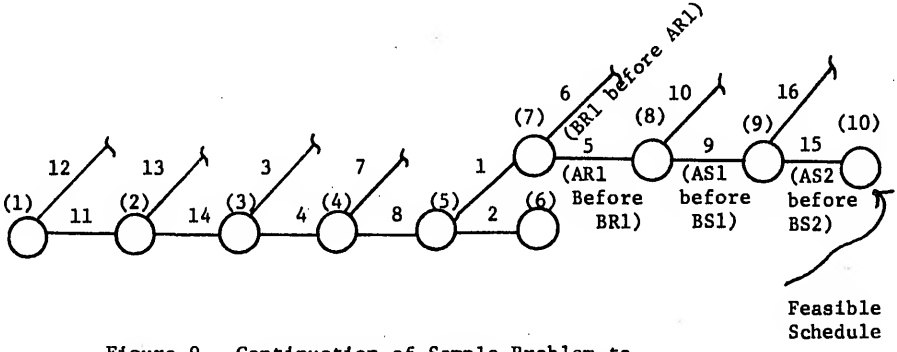


Figure 9. Continuation of Sample Problem to First Feasible Schedule.

We find a solution with the tenth linear program and we need to backtrack. The solution we obtained for the tenth program had variable 16 constrained to zero and variable 15 free. We can backtrack and solve the linear program with variable 15 free and variable 16

constrained to be zero but this is not necessary. The objective function for such a problem would reach 13.89. We only need to start the linear programming algorithm and then we can backtrack again as soon as we obtain an objective function above 10.65. The running list the computer uses would be (12, 11, 13, 14, 3, 4, 7, 8, -1, -2, 6, 5, 10, 9, -16, -15) as we start the linear programming algorithm for the eleventh time and afterwards would be (12, 11, 13, 14, 3, 4, 7, 8, -1, -2, 6, 5, -10, -9).

The program continued to fathom and backtrack in this manner until the problem was solved. Whenever one deals with real numbers there is the possibility of round-off errors. The inverse matrix of the pure scheduling problem has all integers and will not develop errors. Along with the objective function, there is also a vector of real numbers which are used and altered by the linear programs. These numbers can be generated correctly from the inverse matrix but the corrections take time. The program generates the correct values at every solution. Allowances for round-off can be made by estimating the maximum round-off error. Backtracking will not take place from a partial solution unless the partial solution has an objective function greater than an earlier solution plus the maximum error estimate.

Another feature of the program can be seen by the way the given problem was actually solved. Eventually the program fathomed all the solutions with variable 11 free and variable 12 constrained to be zero. Next the solutions had to be fathomed with variable 12 free and variable 11 zero. Variable 12 corresponds to the constraint which forces the second batch of F in the separator to be finished before

the first batch of E begins. This constraint is not consistent with the constraint which would force the second batch of E to begin before the second batch of F. The variable corresponding to the inconsistent constraint must be zero for a feasible solution on this branch of the tree. Also, if the second batch of F begins before the first batch of E we do not need the constraint which would force the second batch of F to begin before the second batch of E. We might as well constrain the variable corresponding to the unnecessary constraint to be zero on a branch if we decide to free one particular variable.

The optimal schedule is given below. It had an objective function of 7.19. Without the program one would have to set up and solve 96 linear programs. The machine solved the problem with a C.P.U. time of 2.89 seconds on an IBM 360/65 computer.

Table 2

Optimal Schedule for Sample Problem

Batch	ER1	ER2	E1S	E2S	FR1	FR2	F1S	F2S
Starting time	5.8	.5	0	4.7	1.2	4.8	2.1	6.8

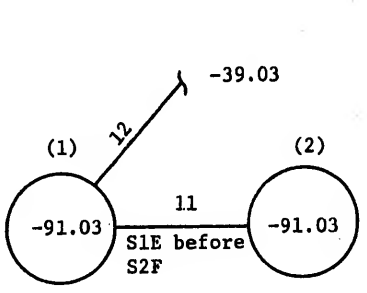
Branch-Bound Enumeration

Earlier the Balas enumeration method was used to schedule operations in a process where the product from two reactors fed a separator. Next we will use branch-bound enumeration to solve the

same problem. Again we will look at how the specific problem was solved using a particular solution procedure.

Once more we solve the problem with none of the either/or constraints applied. Again we use the dual and find that putting variable 12 in the basis would increase the objective function more than putting any other member of the either/or pairs in the basis. Again we want to look at the result of having variable 11 free. Variable 12 corresponds to finishing the second batch of F in the separator before starting the first batch of E. Variable 11 corresponds to finishing the first batch of E in the separator before starting the second batch of F.

The objective function increases to -39.03 when variable 12 is put in the basis. This time the solution procedure will tell the computer to store the number -39.03 and enough information to generate the partial solution with variable 12 in the basis.



Putting variable 12 in basis will increase objective function to -39.03. -39.03 stored in computer along with enough information to regenerate solution.

Optimal schedule with S1E before S2F has objective function of -91.03.

All either/or variables constrained to zero

Figure 10. Flow Chart for First Step in Solving Sample Problem Using a Branch and Bound Enumeration Procedure.

Next the solution procedure looks at putting an either/or variable in the basis in addition to variable 11. Again we find the variable, 13, which increases the objective function the most. Except for the storage of additional information, the branch-bound procedure is the same as the Balas enumeration procedure until the fourth partial solution is found.

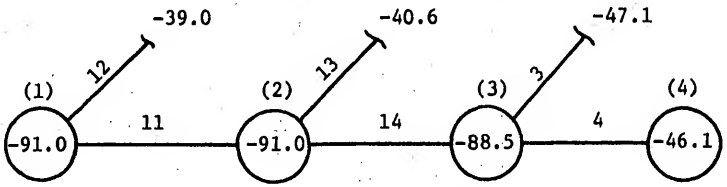


Figure 11. Flow Chart for Branch-Bound Procedure up to Fourth Partial Solution.

Figure 11 shows the results up to the fourth partial solution. Enough information was saved to find the partial solution obtained by freeing variables 11, 14 and 3. The objective function of this partial solution will be larger than -47.13 but it may be smaller than the objective function of the fourth partial solution.

In the branch and bound method, enough information is stored to reproduce the fourth partial solution and the partial solution with variables 11, 14, and 3 free is found.

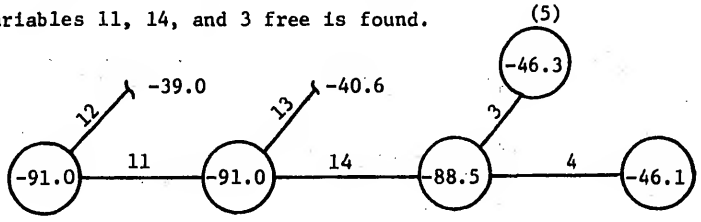


Figure 12. Flow Chart Including Fifth Partial Solution.

The question to be faced next was whether to branch back to the fourth partial solution or to continue working on the fifth partial solution. There is a certain amount of work the computer has to do in order to switch consideration from one partial solution to another. On the other hand, by adding constraints to the fourth partial solution we might find a solution which has an objective function smaller than -46.39. In that case any work done on the fifth partial solution would be wasted effort for no solution derived from the fifth partial solution can have an objective function smaller than -46.39.

Some research is needed on when a solution procedure should switch attention from one partial solution to another. The computer program written uses a number, zb, to make its decisions. If the program is working on a partial solution with an objective function, z, then the program will switch consideration to another partial solution if and only if the other partial solution has an objective function less than z - zb.

The program was run with an arbitrary value of zb = 1.0 so the program switched from the fourth partial solution to the fifth partial solution but did not switch back again until later. If zb is very large a program will spend a large amount of time looking at nonoptimal solutions. zb shouldn't be any smaller than the possible error in the data. One shouldn't try any optimization that can't be justified by the original data.

The branch-bound procedure takes advantage of the structure of the problem as did the earlier method. Eventually the partial solution with variable 12 free had to be investigated. This corresponds

to the second batch of F in the separator being completed before the first batch of E in the separator. The algorithm never considered adding constraints forcing the first batch of F in the separator to be completed before or after the second batch of E.

The linear programming and branching algorithms ignore the ith constraint if the ith member of a vector of logical variables is set free. The computer has to do some work to set up the proper linear program when the partial solution with variable 12 free is investigated. This work is not done until the machine investigates the partial solution. Sometimes information is stored on partial solutions which are never investigated. Any extra work on such partial solutions is unnecessary.

If an incumbent feasible schedule is found with an objective function, zi, then no work need be done on partial solutions whose objective functions are larger than zi. The program which was written stopped work on a partial solution whenever the objective function got larger than such a zi. Partial solutions with large objective functions were removed from storage whenever a new feasible schedule was found.

One advantage of the new branch-bound algorithm is that fewer partial solutions have to be investigated than by the Balas enumeration method. If the number, zb (previous page), is set at zero we will not look at any partial solution whose objective function is larger than the optimal solution. This will happen for the Balas enumeration method only in the lucky case where the optimal solution is the first feasible

solution found. Nineteen partial solutions were investigated using the branch-bound method and thirty-four partial solutions were investigated using the Balas method.

Some extra work was required in storing and retrieving information for the branch-bound method. It took 4.85 seconds for the computer to solve the problem using branch-bound and 2.89 seconds using Balas enumeration. Balas' enumeration had been compiled to produce only 278 lines of output as compared with 391 lines of output for the newer method. Storage of information and output takes some C.P.U. time.

The branch-bound method also required more storage space. 2034 bytes of information had to be in common storage for the branch-bound method and 1518 bytes of information had to be in common storage for the other method. Actually one doesn't know how many partial solutions will have to be stored for the branch-bound method.

A Solution to the Storage Problem

One problem with the branch-bound just developed earlier is that an enormous number of partial solutions have to be stored in a realistic problem. There is no way, short of solving the problem, to tell how many partial solutions will have to be stored.

A large problem might have so many partial solutions that the branch-bound method cannot be used on a particular computer. It is possible to use auxiliary storage devices but retrieval of information becomes slow. On some computers the rate of retrieval of information in core storage is affected by the amount of information stored.

A method which has been programmed is to assign a fixed number

of storage locations for partial solutions. As long as the storage space is not exceeded, the program uses the branch-bound procedure on a scheduling problem. Once the storage space is filled the program completes the best partial solution by fathoming and then returns to the branch-bound method until fathoming has to be used again or the problem is solved.

The same problem was solved that was solved earlier by two different methods. When we used the branch-bound method storage locations for ten partial solutions had to be used.

The problem was run using only the storage space for four partial solutions. When the program had four partial solutions stored it fathomed the partial solution it was working on. Luckily the optimal solution was a result of fathoming this partial solution. 25 partial solutions were investigated as compared to nineteen using pure branch-bound and 34 using pure fathoming. The third solution procedure took 4.19 seconds for 684 lines of output.

Optimization Strategies

In the latter part of Chapter III we looked at the information available on the effect of adding a variable to a linear programming problem. The value of this information can be evaluated now that we've developed some enumeration techniques.

We could use the gradient of the objective function in order to make our scheduling decisions. The gradient is easy to obtain but it does not show how much the objective function will increase if a variable is added to a linear programming problem.

One can determine how much the objective function increases when a given variable, L, is added to the basis. If the objective function will be larger than the objective function for a known schedule then one knows adding L cannot be part of an optimal strategy.

In the example of Chapter III we saw how one decision could have multiple effects. We knew that process E had to be completed before process G was begun. This meant that the decision to begin process F before E also meant that F would begin before G.

We will frequently encounter multiple batches in chemical scheduling problems. If one schedules a batch E before the first of many batches F then one has scheduled E before all of the batches of F. The multiple effects of one decision should justify the effort of finding how much the objective function increases when a variable is added to the basis. Also this lower bound of the objective function makes possible our branch-bound method of enumeration.

In the branch-bound method we switched our attention from one partial solution to another. One would like to be able to reproduce a partial solution without having to solve the linear program formed by the variables used in the partial solution. The partial solution can be reproduced if one has the inverse of the matrix formed from the basis vectors. Storage of an entire matrix for every partial solution is difficult. One could merely store information on which vectors were in the basis. Then one has to invert a matrix in order to reproduce a partial solution. For matrices used in the pure scheduling problem, a simple method for matrix inversion has been developed in Appendix A.

The enumeration methods developed reduced backtracking by finding the vector which would increase the objective function the most. This strategy is especially effective if one has a good incumbent solution. Suppose one examines a partial solution and has an incumbent solution with an objective function of -60. Suppose the partial solution has an objective function of -5. Suppose one has to add three either/or decisions to the partial solution and the effect of adding each vector to the basis is as given below by Figure 13.

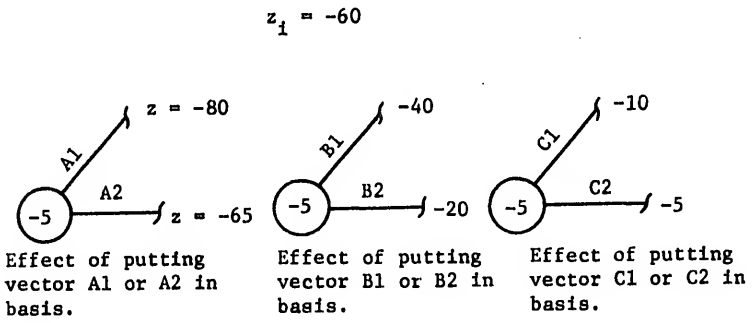


Figure 13. A Partial Solution with Three Remaining Either/Or Choices.

Our enumeration methods would find that adding vector A1 to the basis would cause the objective function to exceed z_i. A1 would be eliminated from consideration and then A2 could be eliminated for the same reason. No further work would have to be done on the partial solution.

Let us look at another example of a partial solution with an incumbent solution.

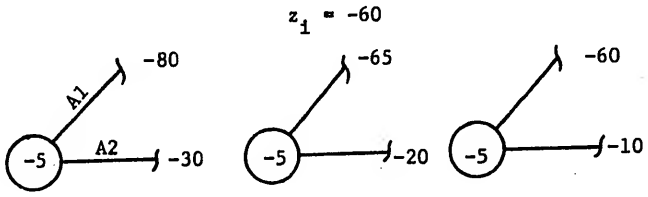


Figure 14. A Partial Solution Where Three Either/Or Choices Can Be Made at Once.

For the example shown in Figure 14 we would eliminate vector A1 from consideration. One need not solve the problem with A2 but could also use the lower bounds on adding B1 and C1 to the problem. With vectors A1, B1, and C1 eliminated, one need only solve the problem with vectors A2, B2, and C2 in order to fathom the partial solution given in Figure 14.

CHAPTER V

ENUMERATION OF MIX/SCHEDULING PROBLEMS

In Chapters I and II a product mix problem for batches was formulated. In order to solve the batch product mix problem one must decide which batches are to be made and how they should be scheduled. All of the enumeration techniques developed in Chapters III and IV also will be used for solving batch product mix problems. Some additional enumeration techniques are developed in this chapter in order to take into account the special structure of batch product mix problems.

In Chapter III scheduling decisions were written as a set of either/or choices. The decisions for batch product mix problems can also be written as a set of either/or choices. Suppose we are to make between two and four batches of a particular type and let x_{ij} be the number of batches to be made. The decisions can be written as either/or choices.

$$\text{either/or} \quad \begin{cases} x_{ij} \geq 4 & (40) \\ x_{ij} \leq 3 & (41) \end{cases}$$

$$\text{either/or} \quad \begin{cases} x_{ij} \geq 3 & (42) \\ x_{ij} \leq 2 & (43) \end{cases}$$

Equations (40) through (43) together with bounding ($2 \leq x_{ij} \leq 4$) express the fact that x_{ij} is integer. Variables for the number of batches made over a time period are bounded because only a certain number of batches can be made in a fixed time period.

Since integer variables can be expressed as either/or choices the techniques of Chapter III can be used for the batch product mix problem.

In Chapters III and IV one decision would sometimes have multiple consequences. This feature is also present for variables such as x_{ij} since the decision to use Equation (40) is inconsistent with the decision to use Equation (43). Equation (40) also makes Equation (42) redundant. It is obvious that if x_{ij} is greater than an integer n that it is greater than an integer m if n > m. This fact has been used before in solving integer programs.[13,17] Actually Trotter[17] used only this fact in addition to the enumeration schemes of Balas[14] and found he could solve the same integer problems faster than with the Balas enumeration scheme.

Trotter's[17] programs are written for integer variables so he would have to add integer variables to a problem in order to express either/or choices such as Equations (11). Additional variables make a problem more complex. Our problems are solved in the dual using the revised simplex method. In the dual Equations (40) through (43) would be represented as variables. In the dual, the choice corresponding to Equation (40) will imply that the variables corresponding to Equations (41) through (43) are constrained to zero. By using the revised simplex method, one can ignore any information about variables which are constrained to zero.

The batch product mix problem optimizes operations over a fixed time period τ. In Chapter II (see Equations (9) and (10) and discussion) we showed that the mathematically correct objective function

could be obtained if the decision not to produce a particular batch is expressed with this particular batch having a starting time of τ.

It is desirable to have a method where one either/or decision can imply another. This feature is implemented in the optimization methods of Chapters III and IV. If one decision can imply another, it is easy to integrate the scheduling and production rate decisions of batch product mix problems. Since one decision can imply another, one only needs two either/or choices to integrate Equations (40) through (43) with scheduling decisions

$$t_{ij4} + p_{ij4} \leq \tau \tag{44}$$

either/or

$$t_{ij4} \geq \tau \tag{45}$$

$$t_{ij3} + p_{ij3} \leq t_{ij4} \tag{46}$$

either/or

$$t_{ij3} \geq \tau \tag{47}$$

Equations (44) through (47) together with bounding ($0 \leq t_{ij3} \leq t_{ij4} \leq \tau$) express the equivalent scheduling decisions as the production decisions of Equations (40) through (43). Equation (40) should imply Equation (44) and vice versa since batch 4 of i in j has to be scheduled in the time period if it is to be made. Equations (42) and (46) imply each other since the third batch of i in j is finished before the fourth batch if it is to be made.

In Chapter III we struggled with formulating scheduling constraints correctly so that the batch product mix problem could be written as an integer programming problem. The scheduling constraints come from the fact that a batch production unit can only operate on one batch at a time.

For one of the batch product mix problems a product, W, could be made in two separators, S2 and S3. SW2 represented the number of batches of W made in S2 and SW3 represented the number of batches of W made in S3. A batch of S2 produced 12.74 pounds of W and a batch of S3 produced 13.44 pounds. At least 23 pounds of W were needed over a fixed time period. A simple material balance is given by Equation (48).

$$12.74\ SW2 + 13.44\ SW3 \geq 23.0 \tag{48}$$

SW2 and SW3 are integer for any feasible product mix since we only consider an integer number of batches. For integers SW2 and SW3 Equation (43) is equivalent to Equation (49).

$$SW2 + SW3 \geq 2 \tag{49}$$

Any integer solution which satisfies Equation (49) also satisfies Equation (45). Any integer solution excluded by Equation (45) is also excluded by Equation (49). Equation (49) was used instead of Equation (48) since Equation (49) can be used to eliminate some either/or choices from consideration. For example if the decision is made that $SW2 \leq 0$ then Equation (49) will force $SW3 \geq 2$. With $SW3 \geq 2$ Equations (50) and (51) can be eliminated from consideration

$$\text{either/or} \quad \begin{matrix} SW3 \geq 2 \\ SW3 \leq 1 \end{matrix} \tag{50}$$

$$\text{either/or} \quad \begin{matrix} SW3 \geq 1 \\ SW3 \leq 0 \end{matrix} \tag{51}$$

Equations similar to (48) were simplified wherever possible in order to eliminate some either/or choices from consideration. The enumeration schemes were implemented on a computer and the computer was able to store the information that the decision $SW2 \geq 0$ eliminates Equations (50) and (51) from consideration. Storage and retrieval of information takes time. One has to decide how much effort should be expended in order to eliminate some either/or choices from consideration. Trotter[16] has shown that some effort should be expended to eliminate either/or choices.

So far we have only shown how one decision can eliminate some either/or choices from consideration. Sometimes two decisions can imply a third decision. For example, look at Equation (52).

$$D + E + F \geq 3 \tag{52}$$

If D, E, and F are integers the decisions $E \leq 0$ and $F \leq 0$ imply that $D \geq 3$. In other words, the decision $D \leq 2$ is eliminated from consideration. General programs were written so that any number of decisions could be made to eliminate an either/or choice from consideration. For the smaller of the two example problems formulated, the author tried to find all instances where one or two decisions would imply a third decision. For the larger of the two problems, the author only tried to find those instances where one decision would imply another. This information is fed into the written computer programs as data. A subprogram could be written to generate this data from equations similar to Equations (48) and (52).

The smaller of the two problems formulated involved producing

and scheduling the same type batches as were scheduled in Chapter IV (see Figure 5). The process involved two reactors and a separator. Two products, E and F, could be processed in either reactor and then processed through the separator. The batch times and inventory costs are the same as those given in Table 1. Again we are to optimize operations over a time period of 9.5 but this time we have to decide both how many batches are to be made and how to schedule them.

Once more we assumed that there was enough beginning inventory to feed either a batch of E or F to the separator. Feed for any additional batches in the separator had to be made by the reactors.

It is not too difficult to schedule sufficient inventory for the second batch of E in the separator. One merely needs constraints to say that one of the batches in the reactor must proceed the second batch in the separator.

$$t_{ER11} + 3.7 \leq t_{ES2} \tag{53}$$

$$t_{ER21} + 4.2 \leq t_{ES2} \tag{54}$$

It is more difficult to schedule the third batch of E in S. The problem assumed two batches had to be produced in a reactor before the second batch could be made in the separator. One really has four choices for producing feed for the third batch of E in S.

$$\text{either} \quad t_{ER11} + 3.7 \leq t_{ES3} \tag{55}$$

$$\text{/or} \quad t_{ER12} + 3.7 \leq t_{ES3} \tag{56}$$

$$\text{/or} \quad t_{ER21} + 4.2 \leq t_{ES3} \tag{57}$$

$$\text{/or} \quad t_{ER21} + 4.2 \leq t_{ES3} \tag{58}$$

Feed for the third batch of A could be produced by either the first or second batch of either reactor. Again we need the ability for one decision to imply another. The solution procedure could not choose both Equations (53) and (55) since the same batch from a reactor could not be used to feed two batches in the separator. Similar equations to Equations (53) through (55) were formulated for chemical B.

The costs used in this small batch product mix problem are given in Table 3. A total of 121 equations were written to formulate this fairly simple batch scheduling problem. One hundred of the equations were of the either/or type. The optimal schedule and an interpretation of the results are given in Chapter VI.

Table 3. Batch Costs for Sample Batch Product Mix Problem

Batch	E in R1	E in R2	E in S	F in R1	F in R2	F in S
Cost	27.0	29.0	-197.0	27.0	29.0	-243.0

A large batch product mix problem was formulated to incorporate the features of a realistic chemical engineering problem. The problem included recycle streams due dates for some materials, different starting inventories and available time for as many as 20 batches on one machine. The flow sheet for the large batch product mix problem is given by Figure 15. Information on the batch characteristics and beginning inventories is given by Tables 4 and 5. The inventory costs are the difference between the cost of storing product minus the

Table 4. Batch Characteristics for Large Batch Product Mix Problem

Process	Unit	Batch Time	Cleanout Time	Batch Cost	Inventory Cost
1	R1	2.5	0.0	525	-9.70
1	R2	3.0	0.0	500	-8.82
1	S1	3.2	0.0	225	.52
1	S2	3.1	0.0	225	.14
2	S1	4.1	0.0	-873	-2.25
2	S3	3.9	0.0	-1109	-2.62
3	R2	3.0	0.5	135	-.88
3	R3	2.5	0.5	150	-1.10
3	S2	1.1	0.0	-2369	-.96
3	S3	1.0	0.0	-2509	-.68
4	R1	2.1	0.0	90	-2.94
4	R3	1.8	0.0	100	-3.23
4	S1	6.2	0.0	-501	-5.14
4	S3	6.6	0.0	-560	-8.00
5	S1	5.1	0.0	-201	-1.54
5	S2	5.4	0.0	-129	-.68

Table 5. Chemical Inventories and Batch Sizes

Chemical	Beginning Inventory	Consumers		Producers	
		Unit	Amount	Unit	Amount
F	Not limited	R11 R21	10 9		
L	0.0	S11 S21	12 14	R11 R21	9.7 8.82
E	25.0	S12 S32 R23 R33	9 11 4 5	S11 S21	6.60 8.40
Q	37.0	R14 R34	7 8	S11 S21 S14 S34 S15 S25	3.84 4.20 1.00 1.05 2.80 3.75
D	32.0	S15 S25	14 15	S12 S32 S15 S25	.81 1.32 5.60 5.25
K	0.0	S23 S33	13 14	R23 R33	4.4 5.5
U	0.0			S12 S32	8.10 9.57
W	0.0			S23 S33	12.74 13.44
G	25.0	S14 S34	20 21	R14 R34 S14 S34	6.65 7.52 2.40 2.94
H				S14 S34	14.00 15.54

Table 6. Feasible Schedule Found for Batch Product Mix Problem

Batch	Batch Number	Starting Time
R11	1	3.4
	2	5.9
	3	8.8
R23	1	0
	2	3.0
	3	6.0
	4	9.0
	5	13.0
	6	16.0
R34	1	2.2
	2	4.0
	3	5.8
	4	10.2
	5	12.0
R33	1	16.5
S14	1	7.6
	2	13.8
S15	1	2.5
S21	1	8.4
	2	11.5
S25	1	14.6
S33	1	12.0
	2	19.6
S32	1	15.1
S34	1	5.4

cost of storing feed. Operations were to be optimized over a time period of 20. A shipment of 10 units of W was due at 13.0 and a shipment of 13 units of W was due at 20. A shipment of 40 units of H was due at the final time but no additional H could be sold. The author found that it took 548 either/or equations to force the inventory of L positive over the whole time period regardless of how many batches were made. For this reason the author left out constraints to keep inventory positive over the whole time period.

The approximation used assumes that some inventory could be bought if needed and resold at the end of the period with negligible cost.

Both of the batch product mix problems were solved using the Balas enumeration procedure of Chapter IV. The enumeration procedure works better with a good incumbent solution and the author found a feasible schedule to the large batch product mix schedule. This feasible schedule is given in Table 6. The feasible schedule was used as an incumbent solution which increased the efficiency of the solution procedure. The solution procedure looks for objective functions which were at least $10 better than the incumbent solution. A discussion of results is found in Chapter VI. The results are compared with work by other authors in order to make them more meaningful. This comparison is done in the first section of Chapter VI.

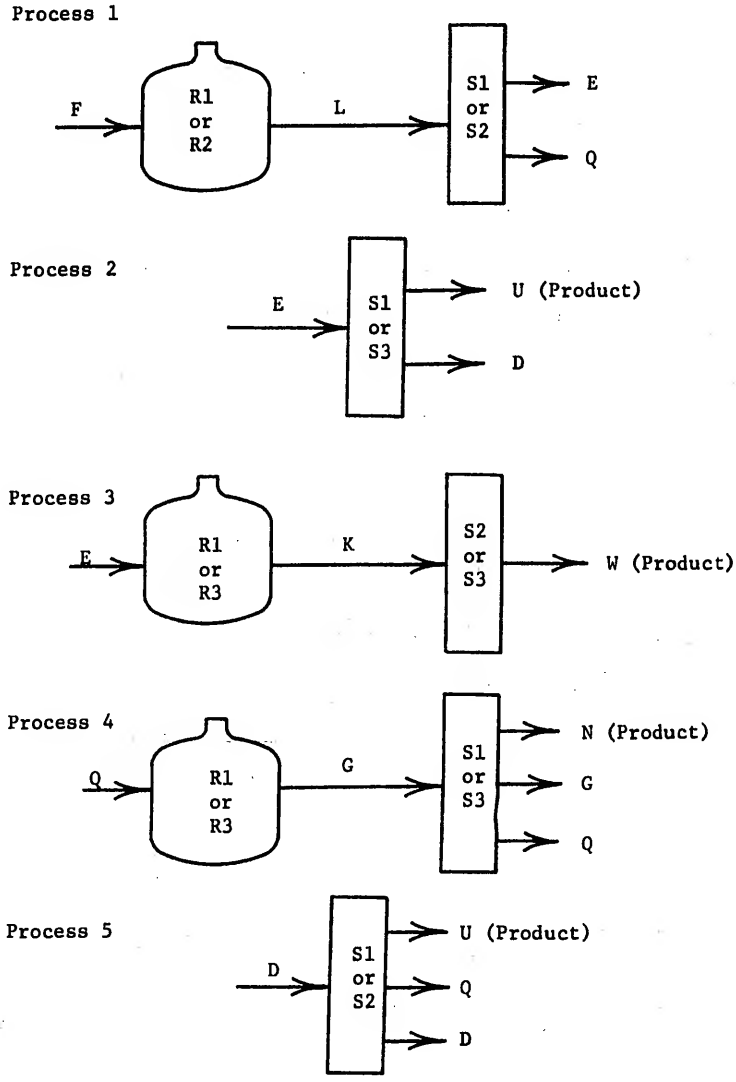


Figure 15. Second Batch Product Mix Problem.

CHAPTER VI

RESULTS, CONCLUSIONS, AND RECOMMENDATIONS

The purpose of this work was to find a method of scheduling batch chemical operations. The method found takes large amounts of computer time if one wants to find optimal schedules for normal sized chemical scheduling problems. The method can find near optimal solutions with a moderate amount of computing time and has the advantage that the limit of suboptimality is expressed in dollars. That is the program can find a solution such that the objective function is within at least a certain number of dollars of being the optimal solution.

Solution Efficiency

The quality of the new scheduling method can be seen by comparing results with other methods of solving discrete optimzation problems. Table 7 shows published results of solving integer programming problems by a number of methods.

As mentioned earlier, Trotter[17] uses the Balas enumeration method but reduces the size of the problem by using integer variables instead of 0-1 variables. Trotter reduces the size of a problem by realizing that $x \geq 4$ implies $x \geq 3,2....$ Our program can do that in addition to reducing the size of a problem by other methods.

Trotter's programs only solve problems where all of the variables are integer. These problems are simpler than scheduling problems because there is a method of solving pure integer problems without using linear programming. One has to check each constraint for feasibility and use a method which guarantees feasibility while optimizing the problem.

Table 7. Computational Results Using Several Integer Programming Methods

Author	Number	Number Constraints	Number Variables	Number of 0-1 Variables	Iterations First Solution	Iterations Optimality Solution	Computer Time
Trotter[17]	8	10	12	30	-	127	.3
	8	10	12	30	-	1430	4.2
	12	15	15	45	-	399	1.5
	12	15	15	45	-	409	16.4
	14	31	31	124	-	12920	52.9
	14	31	31	124	-	14606	>1024
Shareshian[18]	3	28	35	35	45	68	16.8
	4	12	44	44	1148	1255	85.8
Driebeck[19]	2	7	21	21	29	29	-
	6	53	15	15	13	38	-
Greenberg[16]	4	12	8	12	-	19	143
Dzielinski[20]				<16			354
				<16			540
Trauth[21]	4	15	15	15	-	351	26.2
	4	15	15	15	-	1144	66.5
	9	50	15	15	-	752	75.1
	9	50	15	15	-	6758	633
Swanson	1	16	8	8		34	2.9
	1	16	8	8		19	4.9
	1	16	8	8		25	4.2
	2	121	21	34	14	95	35
	3	923	93	270	214	-	>5400

Trotter uses more than one method to optimize his problem and the best and worst results are listed in the table. Trotter's solution methods are highly dependent on how many constraints are present and he did not solve a problem with as many constraints as two of our problems. Trotter used a CDC 6600 computer for his programs.

Shareshian[18] has presented a program to solve mixed integer problems by the branch-bound method. Unfortunately the program was only for some pure integer problems. Shareshian's results look comparable to ours except for the number of constraints and the way the problems are formulated.

In Chapters III, IV and V we found a method of formulating decisions as either/or choices without using 0-1 variables. Every solution method listed on Table 7 except the method of Greenberg would have to use these 0-1 variables. For the other methods our example problem would have 16 (8 + 8) variables, Problem 2 would have 55 (21 + 34) variables and Problem 3 would have 363 (93 + 270) variables.

Both Shareshian's work and this work solve a number of linear programs in the dual. The difficulty of solving these problems is proportional to the number of variables in the primal. Shareshian's results were calculated on an IBM 360/50 computer.

Shareshian's program would also have to add an additional set of variables to integrate production decisions with the scheduling decisions of the second and third sample problems. In Chapter II we first saw that some scheduling decisions do not apply if one decides not to make the batches scheduled. It would take 32 additional variables to write the second sample problem for a normal integer

programming algorithm and 116 additional variables to write the third problem. Shareshian's test problems were solved on an IBM 360/50.

Driebeck[19] developed an algorithm for mixed integer problems that is somewhat similar to the method developed here. Driebeck uses a lower bound on the cost of making a decision and uses this bound to determine his solution strategy. Driebeck also has to solve a new linear programming problem at each iteration as we did. Driebeck's method does have a very bad flaw. Driebeck uses a table which initially contains information on every possible integer solution. Driebeck's table has a vector for each integer solution. The 2^{540} vectors necessary for our third sample problem would certainly be too much for any present day computer to handle.

Greenberg[16] was the only other author found who also realized that 0-1 variables are unnecessary for scheduling problems. Greenberg's results do not look very good because he used a random method of adding constraints to a problem and he did not take advantage of the structure of the problem as this author did. Greenberg used a G.E. 265 Time-Sharing System.

Dzielinski[20] found a method of approximating a restricted class of scheduling problems by a linear program. Dzielinski formed a linear program which contained a cost and set of constraints for every possible schedule. Dzielinski, like Greenberg, would run into serious problems with a scheduling problem like sample problems two and three. Dzielinski did not write all his constraints but used the decomposition principle to handle the problem. Dzielinski's iterations are each simple linear programming iterations but he has to perform a huge

number of iterations to solve the program. Dzielinski used an IBM 7090 computer.

Trauth[21] solves some pure integer programming problems using several cutting plane methods. His best and worst results are listed for several problems. His best solution times were on a CDC 3600 computer and the worst were on an IBM 7090 machine.

All of the author's work was done on an IBM 360/65.

Results

The results given in Table 7 indicate that the method developed here for solving scheduling problems is a better method than other integer programming techniques. Unfortunately most of the work on integer problems has been on problems of the size of our second sample problem and realistic chemical engineering problems are at least the size of the third sample problem. The reader should remember that only eight extra 0-1 variables will increase the number of integer solutions by more than one hundred if all the solutions are feasible.

The optimal schedule for the small batch scheduling problem described in Chapter IV is given by Table 8. There are several interesting features of this schedule. For one thing one should note that the product mix result is to produce two batches of each type in the separator. The operating units could produce more F and less E which is more profitable, if there were no inventory costs. A batch product mix problem would get the wrong answer if it ignored inventory costs.

Table 8. Optimal Product Mix and Schedule for Sample Problem 2

Chemical	Unit	Batch	Starting Time
E	R1	1	1.1
F	R2	1	2.2
E	S	1	0.0
E	S	2	4.8
F	S	1	2.1
F	S	2	6.9

There is very little slack time in the separator but there is some slack time in both reactors. This slack time could be used for building up inventory for future operations.

Suppose one decides to use the slack time in R1 to produce a batch of E and the slack time in R2 to start producing product F. The next production period would have a batch of E for feed to the separator and a batch of F for the separator 2.2 hours after the start of the second time period. If the batch of F was available by 2.1 the optimal schedule for the second time period would be the same as the schedule for the first time period. This schedule for two time periods is given by Table 9. The result on Table 9 may not be the optimal product mix and schedule over two time periods but it is feasible for the beginning inventory and the schedule can be duplicated for any number of time periods. Thirty-five seconds' computer time is certainly a cheap way to get a feasible schedule for any number of time

Table 9. Schedule for Two Time Periods

Chemical	Unit	Batch	Starting Time
E	R1	1	1.1
E	R1	2	5.8
E	R1	3	1.1 + 9.5
E	R1	4	5.8 + 9.5
F	R2	1	2.1
F	R2	2	6.8
F	R2	3	2.1 + 9.5
F	R2	4	6.8 + 9.5
E	S	1	0.0
E	S	2	4.8
E	S	3	0.0 + 9.5
E	S	4	4.8 + 9.5
F	F	1	2.1
F	F	2	6.9
F	F	3	2.1 + 9.5
F	F	4	6.9 + 2.1

periods. The schedule would only have a 1% slack time in the separator.

There is no easy way to account for the inventory needs of a future period when optimizing operations over an earlier period. One could form a scheduling problem and use slack time by giving some value to producing intermediate inventory during this slack time. The optimization techniques of this paper could solve such a problem. The difficulty lies in assigning a value for intermediate inventory. Actually it costs to store intermediate inventory and creation of intermediate inventory is the wrong policy if there is no need for it in the future.

Unfortunately the author did not have enough computer time to solve the large batch scheduling problem of Chapter V. The author attempted to find a schedule that was within $10 of being the optimal schedule. This objective turned out to be very difficult to achieve. If one looks at Tables 4 and 5, $10 may seem like a strong constraint on the solution when some batches are worth over $2000. The author chose $10 because some batches only cost $.70 an hour to store in inventory.

Two weeks was spent finding a schedule for the third sample problem. The schedule that was found made a profit of $4925 which can be compared to the optimal profit of $8500 for the continuous solution. The computer looked for solutions with profits above $4935 and found a schedule with $5449 profit after 22 minutes of work. The computer found another schedule that made $5464 after twenty-seven minutes of computing. The machine had not found a schedule with a profit

of more than $5474 after 90 minutes of computing. The author spent 28 more minutes of computer time trying to prove there was a schedule with an objective function value of at least $5964 ($5464 + $500). The author proved that there was no solution with an objective function value as great as $6964 ($5464 + $1500) by spending an additional 57 minutes of computing time. All computing was done with an IBM 360/65 computer.

Perhaps the branch-bound method would have proved better for solving the large scheduling problem. For the smaller problem of Chapter IV, branch-bound used more computer time than the Balas enumeration method. Perhaps branch-bound is the best method for larger problems.

Of course, now it is easy to see that we should have started with an objective which was easier to obtain than getting within $10 of the optimal solution. The strategy was tested by starting the program over with the objective of getting to within $500 of the optimal schedule. The algorithm first eliminated a decision because of high costs after 17 iterations. With an objective of getting within $10 of the optimal solution the computer did not eliminate a prior decision until after the thirty-third iteration. Since there were 270 decisions to be made eliminating a decision after 17 decisions eliminates 2^{270-17} possible schedules and waiting until the thirty-third decision eliminates 2^{270-33} schedules.

There is another method of improving the efficiency of the solution procedure. Today's computers cannot look at 2^{270} possible schedules in any reasonable length of time. This solution procedure was

developed so that a large number of schedules could be eliminated from consideration. The cases were found where one or more decisions implied other decisions and this was included as input data to the computer. Needless to say the last sample problem was too complex for one person to find all the cases where one or more decisions imply other decisions. However, one could write a computer program to find this information.

The author found a number of cases where one decision would imply another and gave this information to the machine. If the machine had more information then the solution would probably have been obtained more quickly. One set of poor decisions is seen in Table 10. The computer made these decisions while trying to find the first schedule.

Table 10. A Set of Scheduling Decisions

Unit	Process	Maximum Batches	Minimum Batches
R1	1	3	3
R2	1	1	1
S1	1	0	0
S2	1	3	2
R2	3	5	3
R3	3	7	6

The author looked at the solution output and found that the computer was repeatedly finding that both the possible values of S21 were impossible and then the computer would backtrack. The problem with the

decisions was that there was only enough chemical L produced by process 1 to feed two batches of the separator. Two batches in the separator did not create enough feed for the third process. The computer spent over eight minutes backtracking and fathoming in order to discover the last erroneous decision. Since computer time was at a premium the author gave the computer the information that the last erroneous decision was the decision to make only one batch in reactor 2 of process 1.

The reader should note that eight out of the twenty-two minutes spent in finding the solution was spent in finding information that could have been fed into the machine as data. The author did not attempt to find other poor decisions due to a lack of data but it is probable that other poor decisions were made.

The best schedule found by the computer is given in Table 11. This may not be the optimal schedule but it is better than one found with much difficulty by hand. The reader should note that, as before, we have a schedule which makes some inexpensive product (Process 5). Process 5 also makes some recycle.

Conclusions and Recommendations

The solution method was developed to solve scheduling problems more efficiently than other integer programming methods. The efficiency of solution method depended on information about the consequences of decisions. This information proved very important and future work should involve systematic methods of generating this information.

The solution method had some difficulty solving a realistically

Table 11. Best Schedule Found for Sample Problem 3

Process	Unit	Batch	Starting Time
1	R1	1	12.5
1	R1	2	15.0
1	R1	3	17.5
2	R2	1	17.0
2	S2	1	0.0
2	S2	2	9.3
3	R2	1	5.0
3	R2	2	8.0
3	R2	3	11.0
3	R2	4	14.0
3	R3	1	3.4
3	R3	2	5.9
3	R3	3	8.4
3	R3	4	10.9
3	R3	5	15.7
3	S2	1	8.2
3	S2	2	17.8
3	S2	3	18.9
4	R1	1	6.2
4	R1	2	8.3
4	R1	3	10.4
4	R3	1	13.4
4	R3	2	18.2
4	S1	1	7.6
4	S1	2	13.8
4	S3	1	13.4
5	S1	1	2.5
5	S2	1	12.4

sized chemical batch scheduling problem. A fairly good schedule could be obtained with a limited amount of computer time.

Future work should be involved with solving realistically sized problems instead of the small problems, which were the focus of attention in the past.

Of interest to chemical engineers is the result that some batch product mix problems should produce a wider variety of products than they would if inventory costs were not included. It is also of interest to note that scheduling considerations cause a moderate amount of slack time chemical batch units. In the past, one might have supposed that this was due to poor performance by chemical operators.

Also, obviously of interest is the fact that we now have a method of determining both how many batches should be produced and how they should be scheduled.

APPENDICES

APPENDIX A

THE INVERSE MATRIX FOR PURE SCHEDULING PROBLEMS

The branch bound enumeration method of Chapter IV involved the storage and retrieval of a number of partial solutions to the optimization problem. Since the revised simplex method is used, an inverse matrix, B^{-1}, is needed in order to reproduce partial solutions. The matrix, B, to be inverted (see Equation (24)) is made up of columns of the matrix A' which defines the dual problem. It is easier to store information on which columns of A' define B than it is to store the entire matrix B^{-1}. The branch-bound algorithm developed here stores information on which columns define B and then invert the matrix B.

Matrix inversion is normally fairly difficult but a simple method is developed here for inversion of the matrix B found in pure scheduling problems. We will first use the method to invert a matrix formed from the first, third and fifth columns of the matrix A' in Equation (21).

$$B = \begin{bmatrix} 0 & -1 & 1 \\ 0 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix} \tag{59}$$

The third component of the first row of B^{-1} must be -1 since the first row of B^{-1} times the first column of B must give 1. Thus we know one component of B^{-1}.

$$B^{-1} = \begin{bmatrix} B_{11}^{-1} & B_{12}^{-1} & -1 \\ B_{21}^{-1} & B_{22}^{-1} & B_{23}^{-1} \\ B_{31}^{-1} & B_{32}^{-1} & B_{33}^{-1} \end{bmatrix} \tag{60}$$

Since the first row of B^{-1} times the second column of B must give zero, the first component of the first row of B^{-1} must be a minus one.

$$B^{-1} = \begin{bmatrix} -1 & B_{12}^{-1} & -1 \\ B_{21}^{-1} & B_{22}^{-1} & B_{23}^{-1} \\ B_{31}^{-1} & B_{32}^{-1} & B_{33}^{-1} \end{bmatrix} \tag{61}$$

Since the first row of B^{-1} times the third column of B must equal zero, B_{12}^{-1} must be -1.

B_{23}^{-1} equals zero because the second row of B^{-1} times the first row yields 0. B_{21}^{-1} equals -1 because the second row of B^{-1} times the second column of B yields one. B_{23}^{-1} must equal -1 since the second row of B^{-1} times the third row of B equals 0. Similar arguments can be made for the values for the third row of B^{-1}.

$$B^{-1} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & 0 \\ 0 & -1 & 0 \end{bmatrix} \tag{62}$$

A good deal of work is involved in inverting this matrix but no more work is involved in finding the first three rows of a much larger inverse matrix.

$$\begin{bmatrix} 0 & 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} = I \tag{63}$$

In Equation (63) I is the identity operator. The first three rows of the first matrix of Equation (63) can be found by the same methods used in finding B^{-1} of Equation (62).

The procedure for finding B^{-1} in Equations (59) through (63) can be used for all pure scheduling problems. Some characteristics of B for pure scheduling problems will be proven in order to show the validity of the matrix inversion procedure.

Structure of B

Here, we are considering only the pure scheduling problem. We know how many batches of each product to produce over a fixed time period. The cost of a particular schedule, z, will be given by Equation (64).

$$k + c \cdot t = z \tag{64}$$

where t_i are the times that a particular batch starts. We will want to minimize $c \cdot t$. Our constraints will be of the form:

$$\begin{aligned} t_i - t_j &\leq -d_1 \\ -t_i + t_j &\leq -d_2 \end{aligned} \qquad \text{either/or} \tag{65}$$

$$t_i \leq k_i \tag{66}$$

where Equations (65) say that batch t_i is to be started either d_1 units of time before t_j or d_2 units of time after t_j. Equation (66) says that batch i has to be started by time k_i.

We will be doing linear programming problems given by Equations (20).

minimize $c't$			maximize $d'w$		
primal problem	such that	$At \geq d$, $t \geq 0$	dual problem	such that	$A'w \leq c$, $w \geq 0$

(20)

where A includes some set of constraints like Equations (60).

We are going to use the revised simplex method of solving the dual problem. First, we will write the dual in Equation (67).

$$\begin{array}{lll} & \max d'w & \\ \text{such that} & A^* \begin{bmatrix} -1 & 0 & \cdots & 0 \\ 0 & -1 & \cdots & 0 \\ & \cdots & & \\ 0 & \cdots & 0 & -1 \end{bmatrix} & \begin{bmatrix} w \\ w_1 \end{bmatrix} = c \quad \begin{bmatrix} w \\ w_1 \end{bmatrix} \geq 0 \end{array} \tag{67}$$

If the matrix in Equation (67) has m rows we will be using m linearly independent columns of Equation (67) at a time. These m columns form a basis matrix B. It will be helpful to know the structure of B^{-1}. Each column of B has one of the forms given by Figure 16.

$$\begin{matrix}0\\ \vdots\\ 0\\ 1\\ \vdots\\ 0\\ -1\\ 0\\ \vdots\end{matrix} \quad \text{or} \quad \begin{matrix}0\\ \vdots\\ 0\\ 1\\ 0\\ \vdots\\ 0\end{matrix} \quad \text{or} \quad \begin{matrix}0\\ \vdots\\ 0\\ -1\\ 0\\ \vdots\\ 0\end{matrix}$$

Figure 16. Three Possible Forms for Columns of B.

A simplified notation will be used in the following analysis. A column of B with all zeros expect a one in the ith row will be written as e_i. A row of B^{-1} with all zeros except a one in the ith column will also be written as e_i. When a proof is given which will hold for e_i or $-e_i$ the notation $(\pm)e_i$ will be used. The notation $\overset{(-)}{+}e_i \overset{(+)}{-} e_j$ means that discussion is valid for either $e_i - e_j$ or $-e_i + e_j$.

Proof 1

The vectors $e_{\sigma_1} - e_{\sigma_2}$, $e_{\sigma_2} - e_{\sigma_3}, \ldots, e_{\sigma_{n-1}} - e_{\sigma_n}$, $e_{\sigma_n} - e_{\sigma_1}$ are linearly dependent.

This can be seen to be true because they can be added together to give 0.

Proof 2

The vectors $\overset{(-)}{+}e_{\sigma_1} \overset{(+)}{-} e_{\sigma_2}, \overset{(-)}{+}e_{\sigma_2} \overset{(+)}{-} e_{\sigma_3}, \ldots, \overset{(-)}{+}e_{\sigma_{n-1}} \overset{(+)}{-} e_{\sigma_n}, \overset{(-)}{+}e_{\sigma_n} \overset{(+)}{-} e_{\sigma_1}$ are linearly dependent.

If we multiply each of these vectors by a plus or minus one, we will have a vector of Proof 1. Let $x_1 = +\ e^{(-)}_{\sigma_1} - e^{(+)}_{\sigma_2}$, $x_2 = +\ e^{(-)}_{\sigma_2} - e^{(+)}_{\sigma_3}$, etc. By Proof 1 we thus have constants α_i (all plus or minus one) such that $\sum_{i=1}^{n} \alpha_i x_i = 0$. By definition the vectors are dependent.

In the following we will be using some proofs and terminology of graph theory. The proofs will be given in terms of our system and hopefully the proofs will be easier to follow when presented in this manner.

Definition

Among the columns of B we will have a path between e_{σ_1} and $e_{\sigma_{n-1}}$ if there are n columns of B of the form

$$+\ e^{(-)}_{\sigma_1} - e^{(+)}_{\sigma_2},\ +\ e^{(-)}_{\sigma_2} - e^{(+)}_{\sigma_3},\ldots,\ +\ e^{(-)}_{\sigma_n} - e^{(+)}_{\sigma_{n+1}}\ .$$

Definition

The path between the vectors of P-2 will be said to form a loop, since there is a path between e_{σ_1} and itself.

Proof 3

Suppose we have a set S, of columns of B of the form; $+\ e^{(-)}_{\sigma_i} - e^{(+)}_{\sigma_j}$. Suppose that there is no loop formed by members of this set. Suppose further that in this set, S, there is no path between e_{σ_k} and e_{σ_ℓ}. We may add the column $+\ e^{(-)}_{\sigma_k} - e^{(+)}_{\ell}$ to the set S and still not have any loops.

There is no path between e_{σ_k} and e_{σ_ℓ} in the set S. Thus with the addition of $+\ e^{(-)}_{\sigma_k} - e^{(+)}_{\sigma_\ell}$ to S there will only be one path between e_{σ_k} and e_{σ_ℓ} and neither e_{σ_k} nor e_{σ_ℓ} can be part of any loop in the new set. Any path not involving e_{σ_k} or e_{σ_ℓ} will not be affected by the new addition to S.

Definition

Suppose we have a set, S, of columns of B of the form; $-\ e^{(+)}_{\sigma_i} + e^{(-)}_{\sigma_j}$. A unit vector, e_σ, found in the set will be called an extreme point of S if it is only found in one column, $+\ e^{(-)}_{\sigma_k} - e^{(+)}_{\sigma_i}$, of S.

Proof 4

Any set, S, of columns of B of the form; $-\ e^{(+)}_{\sigma_i} + e^{(-)}_{\sigma_j}$, without a loop, has an extreme point.

Suppose n unit vectors, e_{σ_ℓ}, are used in the set S. Choose any vector $e_{\sigma'_2}$, used in the set. Either $e_{\sigma'_2}$ is an extreme point of the set or it is not. If $e_{\sigma'_2}$ is an extreme point we are finished. If $e_{\sigma'_2}$ is not an extreme point we can find two columns of S of the form $+\ e^{(-)}_{\sigma'_1} - e^{(+)}_{\sigma'_2}$, $+\ e^{(-)}_{\sigma'_2} - e^{(+)}_{\sigma'_3}$. Either $e_{\sigma'_3}$ is an extreme point of the set or it is not. Suppose we have found ℓ-1 members of the set of the form; $+\ e^{(-)}_{\sigma'_1} - e^{(+)}_{\sigma'_2}, +\ e^{(-)}_{\sigma'_2} - e^{(+)}_{\sigma'_3}, \ldots, +\ e^{(-)}_{\sigma'_{\ell-1}} - e^{(+)}_{\sigma'_\ell}$ where $e_{\sigma'_2}$ through $e_{\sigma'_\ell}$ are not extreme points. Since $e_{\sigma'_\ell}$ is not an extreme point there must be column of S of the form $+\ e^{(-)}_{\sigma'_\ell} - e^{(+)}_{\sigma'_{\ell+1}}$.

$e_{\sigma'_{\ell+1}}$ cannot belong to the set $e_{\sigma'_1}, e_{\sigma'_2}, \ldots, e_{\sigma'_\ell}$ or we would form a loop. Suppose we eventually find that S consists of a path through n vectors without forming a loop and $e_{\sigma'_2}$ through $e_{\sigma'_{n-1}}$ are not extreme points. We would have the columns, $+^{(-)}_{e_{\sigma'_1}} +^{(-)}_{e_{\sigma'_2}}, +^{(-)}_{e_{\sigma'_2}} -^{(+)}_{e_{\sigma'_3}}, \ldots, +^{(-)}_{e_{\sigma'_{n-1}}} -^{(+)}_{e_{\sigma'_n}}$, and $e_{\sigma'_1}$ through $e_{\sigma'_n}$ are the only unit vectors forming the set. If we have another column of the set of the form; $+^{(-)}_{e_{\sigma'_n}} -^{(+)}_{e_{\sigma'_i}}$, we will have a loop. There will thus be only one column of S of the form, $+^{(-)}_{e_{\sigma'_n}} -^{(+)}_{e_{\sigma'_i}}$, and $e_{\sigma'_n}$ is an extreme point.

Proof 5

No more than n-1 independent vectors of the form $+^{(-)}_{e_{\sigma_i}} -^{(+)}_{e_{\sigma_j}}$ can be formed from n unit vectors e_{σ_i}.

Now Proof 4 is true for n=2 because there can be only one independent vector of type $+^{(-)}_{e_{\sigma_1}} -^{(+)}_{e_{\sigma_2}}$. Suppose Proof 4 is true for $n=\ell$. Suppose we have a set S, of independent vectors of form $+^{(-)}_{e_{\sigma_i}} -^{(+)}_{e_{\sigma_j}}$ formed from $\ell+1$ vectors e_i. From Proof 2 the vectors, $+^{(-)}_{e_{\sigma_i}} -^{(+)}_{e_{\sigma'_j}}$, of S cannot be independent if there is a loop. From Proof 3, if there is no loop, there must be an extreme point, e_{σ_k}. Elimination of the vector $+^{(-)}_{e_{\sigma_i}} -^{(+)}_{e_{\sigma_k}}$ from the set S will create a set formed from only ℓ unit vectors e_{σ_i}, $i \neq k$, which can have at most $\ell-1$ vectors of the form $+^{(-)}_{e_{\sigma_i}} -^{(+)}_{e_{\sigma_j}}$. Our set formed from $\ell+1$ unit vectors must therefore have at most $(\ell-1) + 1 = \ell$ vectors. Q.E.D.

Proof 6

If B has a column of the form $e_{\sigma'_1} - e_{\sigma'_2}$, then there must be a path between either $e_{\sigma'_1}$ or $e_{\sigma'_2}$ and some vector $e_{\sigma'_n}$ where a column of B is of the form $-\overset{(+)}{e}_{\sigma'_n}$.

For this proof one has to remember that B has m rows and m independent columns which are either of the form $-\overset{(+)}{e}_i$ or $+\overset{(-)}{e}_i - \overset{(+)}{e}_j$. Proof 5 will be proved by contradiction. We will prove that if Proof 5 were false then the columns of B would be linearly dependent.

Suppose B has a column of the form $e_{\sigma'_1} - e_{\sigma'_2}$ and let S be the set of all vectors $e_{\sigma'_i}$ where there is either a path between $e_{\sigma'_i}$ and $e_{\sigma'_1}$ or a path between $e_{\sigma'_i}$ and $e_{\sigma'_2}$. Suppose Proof 5 is false. Then there is no column of B of the form $-\overset{(+)}{e}_{\sigma'_i}$ where $e_{\sigma'_i}$ is a member of S.

Let S_c be the set of all columns of B of the form $+\overset{(-)}{e}_{\sigma'_i} - \overset{(+)}{e}_{\sigma'_j}$ where $e_{\sigma'_i}$ and $e_{\sigma'_j}$ are in S. Let n be the number of vectors, $e_{\sigma'_i}$, in S and n_c be the number of columns of B in S_c. Either $n_c < n$ or $n_c \geq n$.

If $n_c < n$ then the $m - n_c$ columns of B which are not in S_c are generated by the m-n vectors e_i which are not in S.

There are m unit vectors e_i in the vector space we are working with. The $m-n_c$ columns of B which are not in S_c are generated by the m-n vectors, e_i, which are not in S.

By Proof 5 if $n_c \geq n$ then the columns of S_c are dependent.

If $n_c < n$ then the $m-n_c$ columns of B which are not in S_c are dependent because there are generated by only m-n vectors e_i. Q.E.D.

Proof 7

If the columns of B were of the form $-\overset{(+)}{e_{\sigma_1}}, +\overset{(-)}{e_{\sigma_1}} -\overset{(+)}{e_{\sigma_2}}, +\overset{(-)}{e_{\sigma_2}} -\overset{(+)}{e_{\sigma_3}},\ldots, +\overset{(-)}{e_{\sigma_{n-1}}} -\overset{(+)}{e_{\sigma_n}}, +\overset{(-)}{e_{\sigma_n}}$ they would be linearly dependent.

Here we have n+1 columns of B generated by n unit vectors. No more than n independent columns can be generated by n unit vectors so we have that these n+1 columns must be dependent.

Generation of the Inverse Matrix

Now that we have shown the characteristics of B if it has independent columns, it is not too hard to deduce the properties of B^{-1}. We can write each row of B^{-1} as $\sum \alpha_i e_i$.

The inverse matrix will be generated one row at a time. The characteristics of the jth row of B^{-1} depend on the characteristics of the jth column of B. A column of B can have either one or two nonzero components and we will see how to generate a row, j, of B^{-1} depending on the properties of the corresponding jth column of B.

The jth Column of B Has Two Nonzero Components

Here the jth column of B, b_j, is given by Equation (63) and we want to find the jth row of B^{-1}.

$$b_j = \overset{(-)}{+ e_\alpha} \overset{(+)}{- e_\beta} \tag{68}$$

First we find all of the vectors, $e_{\sigma'_i}$, where the columns of B form a path between $e_{\sigma'_i}$ and e_α. By Proof 6 there is a column, k, of B of the form $\overset{(+)}{- e_{\sigma'_k}}$. By Proof 7 there is only one such column, k.

The jth row of B^{-1}, b_j^{-1} is given by Equation (66) since the jth row of B^{-1} timea the kth row of B must be zero.

$$b_j^{-1} = \sum_{i \neq \sigma_k} \gamma_i e_i \tag{69}$$

Next we find the path between e_{σ_k} and e_α. There can be two typea of pathe, P1 and P2, between an extreme point and e_α. Theae two types of paths are given by Equations (70) and (71).

$$P1 = \overset{(+)}{-}\, e_{\sigma_x}, \overset{(-)}{+}\, e_{\sigma_x} - e_{\sigma_{x_1}}, \overset{(+)}{+}\, e_{\sigma_{x_1}} - e_{\sigma_{x_2}}, \ldots, \overset{(-)}{+}\, e_{\sigma_{x_n}}$$
$$\overset{(+)}{-}\, e_\alpha, \overset{(-)}{+}\, e_\alpha - e_\beta \tag{70}$$

$$P2 = \overset{(+)}{-}\, e_{\sigma_x}, \overset{(-)}{+}\, e_{\sigma_x} - e_{\sigma_{x_1}}, \overset{(+)}{+}\, e_{\sigma_{x_1}} - e_{\sigma_{x_2}}, \ldots, \overset{(-)}{+}\, e_{\sigma_{x_n}}$$
$$\overset{(+)}{-}\, e_\beta, \overset{(-)}{+}\, e_\alpha - e_\beta \tag{71}$$

If the extreme point e_{σ_k} of Equation (69) belongs to a path of type P1 then γ_i (Equation (69)) is zero if e_i belongs to a path of type P1 and γ_i is $\overset{(+)}{-}$ 1 if e_i belongs to a path of type P2. If the extreme point e_{σ_k} of Equation (69) belongs to a path of type P2 then γ_i is zero if e_i belongs to a path of type P2 and γ_i is $\overset{(+)}{-}$ 1 if e_i belongs to a path of type P1. γ_i is zero if e_i does not belong to a path of type P1 or P2.

The jth Column of B Has One Nonzero Component

Here the jth column of B, b_j, is given by Equation (67) and

we want to find the jth row of B^{-1}.

$$b_j = \overset{(+)}{-} e_\alpha \tag{72}$$

Since the jth row of B^{-1} times the jth column of B is one, the jth row of B^{-1}, b_j^{-1}, is given by Equation (68).

$$b_j^{-1} = \overset{(+)}{-} e_\alpha + \sum_{i \neq \alpha} \gamma_i e_i \tag{73}$$

The vector $\overset{(+)}{-} e_\alpha$ may be an extreme point of one or more paths, P, formed by the columns of B. If a vector e_{σ_i} is part of any such path P (Equation (74)), then γ_{σ_i} of Equation (73) must be $\overset{(+)}{-} 1$ since the jth row of B^{-1} times any column $i \neq j$ of B must yield zero. If a vector, e_k, does not belong to any path given by Equation (74) then γ_k of Equation (63) must be zero.

$$P = \overset{(-)}{+} e_\alpha \overset{(+)}{-} e_{\sigma_1}, \overset{(-)}{+} e_{\sigma_1} \overset{(+)}{-} e_{\sigma_2}, \ldots, \overset{(-)}{+} e_{\sigma_{n-1}} \overset{(+)}{-} e_{\sigma_n} \tag{74}$$

The inversion procedure developed earlier has to find paths among the columns of B. One needs to find the nonzero components of B in order to find these paths. The programs described later store the location of nonzero components of the columns which make up B. The programs also store the value of the nonzero components. Since the columns have few nonzero components it is easier to store their locations and values than it is to store the entire matrix B.

We have a fairly easy method of generating each row of B^{-1} merely by knowing the structure of B. Therefore, we can store which columns of A' are used in a matrix B and invert B in order to obtain B^{-1}.

APPENDIX B

PROGRAMS

The solution procedures developed earlier were implemented in the Fortran language. The programs used are given in this section.

The Balas enumeration procedure is implemented by subprogram FATHOM. The branch-bound enumeration procedure is implemented by subprogram BRANCH. Subprogram LINEAR performs the linear programming for both enumeration procedures and subprogram COST calculates the cost of freeing a variable for both enumeration procedures.

Both enumeration procedures diminish the size of a problem as a consequence of one decision implying another. Subroutine FATHOM uses subroutine DEMINF and subprogram BRANCH uses subprogram DEMIN.

Flow charts are given for the first few subprograms. Listings are given for the other subprograms. A reader who is interested in understanding and using the programs should first read the description of the flow charts.

The major variables are found in these flow charts. Hopefully the flow charts will be useful to future programmers.

Subroutine PIVOT

PIVOT is used along with subroutine LINEAR to solve linear programming problems by the revised simplex method. The revised simplex method solves linear programs by moving from one feasible extreme point to another. We will be solving a linear maximization problem with M constraints and IN variables. An extreme point is characterized by IN-M of the variables having a value of zero. The M nonzero variables are called basis variables.

One should remember that we are solving the dual of a scheduling problem and that a number of dual variables are constrained to zero and will never be candidates to enter the basis. The free variables will have nonzero values when they are in the basis. The dual linear programming problem is given by Equation (75).

$$\begin{aligned} &\text{maximize} && \sum_{i=1}^{IN} -CC(i)x_i \\ &\text{such that} && \sum_{i=1}^{IN} x_i a_i = \begin{matrix} RS(1) \\ RS(2) \\ \cdots \\ RS(M) \end{matrix} \\ & && x_i \geq 0 \quad \text{all } i \end{aligned} \tag{75}$$

Here the a_i are vectors of Equations (22) and (23). The revised simplex method uses an inverse matrix, BI, and a vector, BI, to carry information about the current extreme point. Let x_{B_1}, x_{B_2}, ... x_{B_M} be the basis variables. In this case, BI is the inverse of the matrix formed from the corresponding vector a_{B_i}.

$$BI = [a_{B1}, A_{B2}, \ldots, a_{BM}]^{-1} \tag{76}$$

B1 is a vector caused by multiplying costs for the basis variables times BI.

$$[B1(1), B1(2), \ldots, B1(M)] = [C_{(B1)}, C_{(B2)}, \ldots, C_{(BM)}]BI \tag{77}$$

PIVOT generates new values for BI and B1 when a new variable is put in the basis. The variable K is to be taken out of the basis and the new variable, n, is to be put in the basis and the variable Y expresses a_n in terms of the basis vectors a_{Bi}. The program only needs the vector Y given by Equation (78) to calculate the new feasible solution.

$$\sum_{i=1}^{M} a_{Bi} Y(i) = a_n \tag{78}$$

It can be shown[15] that the formula for the K-th row of the new matrix BI is given by Equation (79).

$$BI(K,j)_{new} = BI(K,j)_{old}/Y(K) \quad \text{All } j \tag{79}$$

All other rows of BI satisfy Equation (80).

$$BI(i,j)_{new} = BI(i,j)_{old} - \frac{Y(i)}{Y(K)} BI(K,j) \tag{80}$$

$$\text{all } i,j \quad i \neq K$$

The formula for the new values of B1 is given by Equation (81).

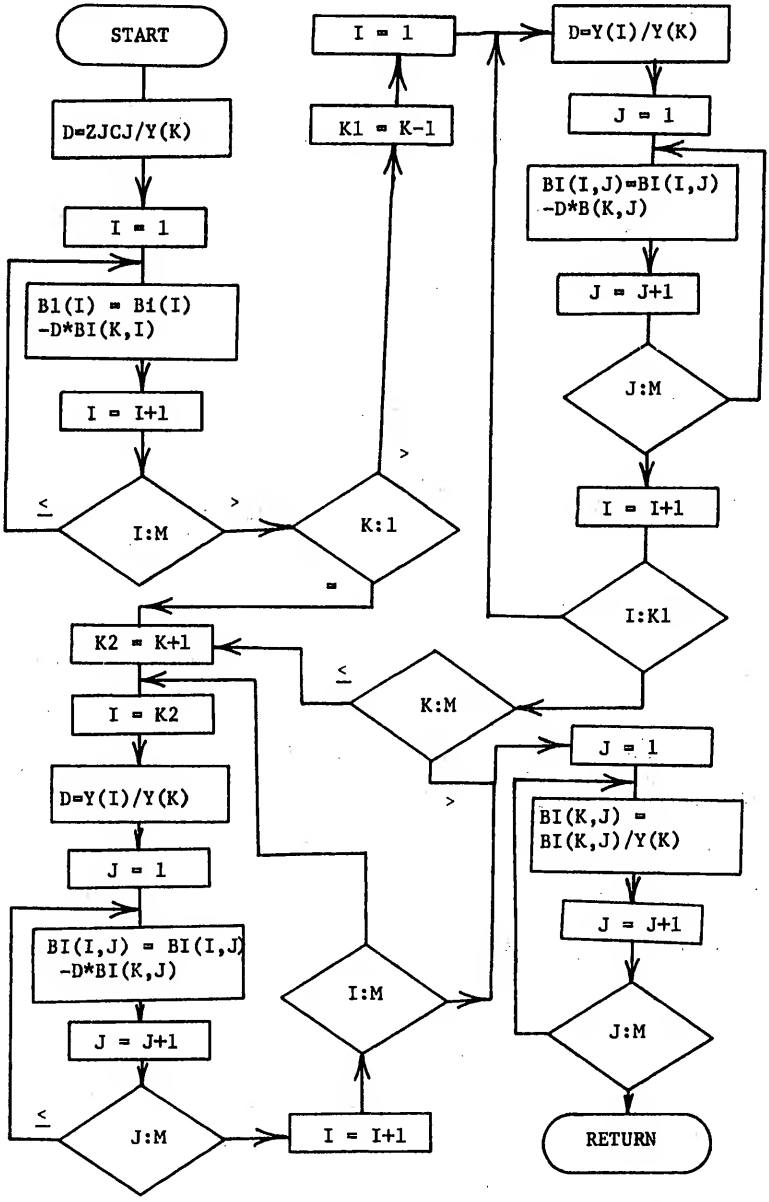


Figure 17. Flow Sheet for Subroutine PIVOT.

$$Bl(i)_{new} = Bl(i)_{old} - \frac{ZJCJ}{Y(K)} BI(K,i) \qquad (81)$$

ZJCJ is a number fed to PIVOT from LINEAR. The formula for ZJCJ is given by Equation (82).

$$ZJCJ = CC(n) + [Bl(1),Bl(2),\ldots,Bl(M)]\ a(n) \qquad (82)$$

PIVOT merely calculates new values of BI and Bl by means of Equations (79), (80) and (81).

Subroutine LINEAR

Subroutine LINEAR solves linear maximization problems by means of the revised simplex method. The maximization problems come from the dual batch scheduling problems of Chapters II-V.

The maximization problems involved will always have IN variables but a subset of these variables will be constrained to zero. A vector, IX, of logical variables indicates which variables are constrained to zero. IX(i) = .FALSE. indicates variable i is not a candidate to enter the basis. Thus if IX(i) = .FALSE. and variable i is not in the basis variable i must be constrained to zero.

The subprogram stores the columns of the original constraint matrix (columns a_i of Equation (75)) in a compact manner in matrix AA. AA(1,j) j=1,...,IN is the number of nonzero components of the jth column of the constraint matrix. If a column j has n_j nonzero components then AA(2,j) through AA(n_j + 1,j) give the location of these nonzero components. The corresponding values of the nonzero components are found in the list AA(n_j + 2,j) through AA($2n_j$ + 1,j).

Iteration Loop 17

Iteration Loop 17 finds the free variable (i.e., variable not constrained to zero) which should enter the basis. The variable which has the most negative value of ZJCJ defined in Equation (82) is chosen to be put in the basis. ZJCJ is actually a derivative of the objective function as one moves along an "edge" of the constraint set from one extreme point to another. Hadley[15] gives some reasons for using ZJCJ in order to make decisions. For one thing, it doesn't take much work to find ZJCJ compared to other information one could calculate.

If a variable, I1, is free and not in the basis (indicated by IX(I1) = .true.), ZJCJ is calculated by Equation (77). N1 = AA(1,I1) + 1 is the number of nonzero terms in column I1 of the original constraint matrix. For each $2 \leq I \leq N1$, N2 = AA(I,I1) is the location of a nonzero component of column I1 of the constraint matrix and the value of the nonzero component is AA(I + N1-1, I1).

Iteration loop 17 finds ZJCJ for each free variable I1 and stores the most negative value of ZJCJ in the storage location ZJOPT. Variable L3 will be the free variable with the most negative value of ZJCJ.

If there is no variable with a negative ZJCJ we go to 10, otherwise we proceed to iteration loop 7.

Iteration Loop 7

Iteration loop 17 decides that variable L3 should be put in the basis. The purpose of iteration loop 7 is to calculate the vector

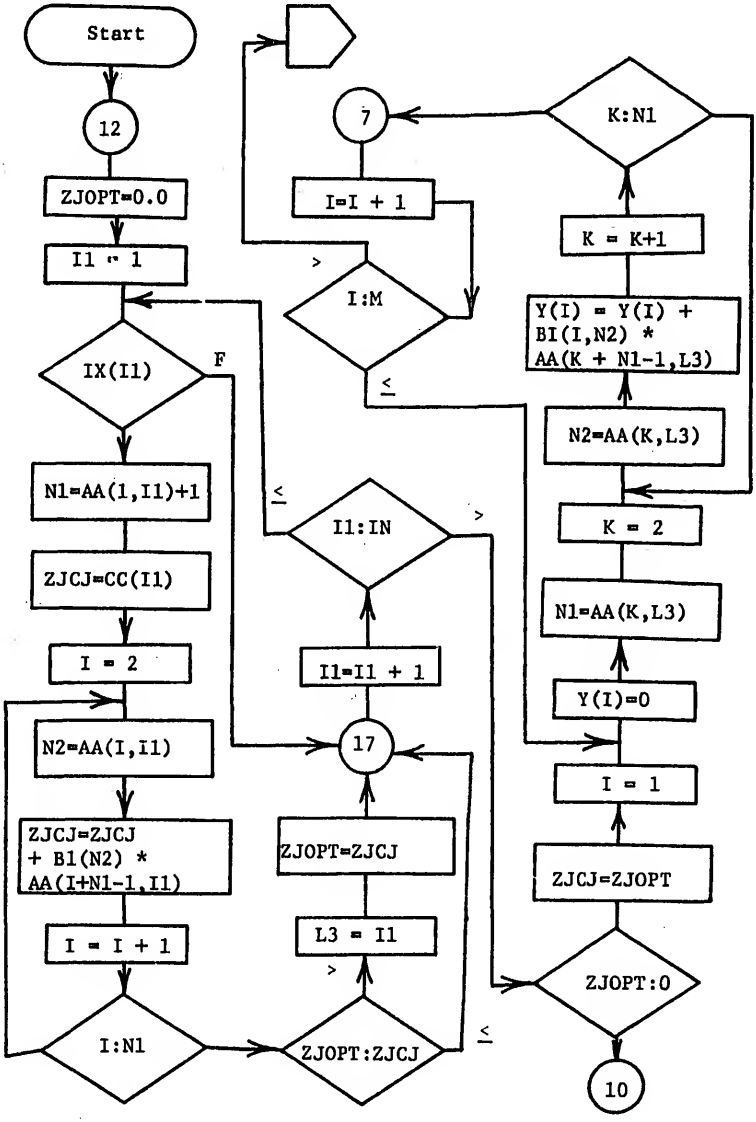


Figure 18. Flow Sheet for Subroutine LINEAR.

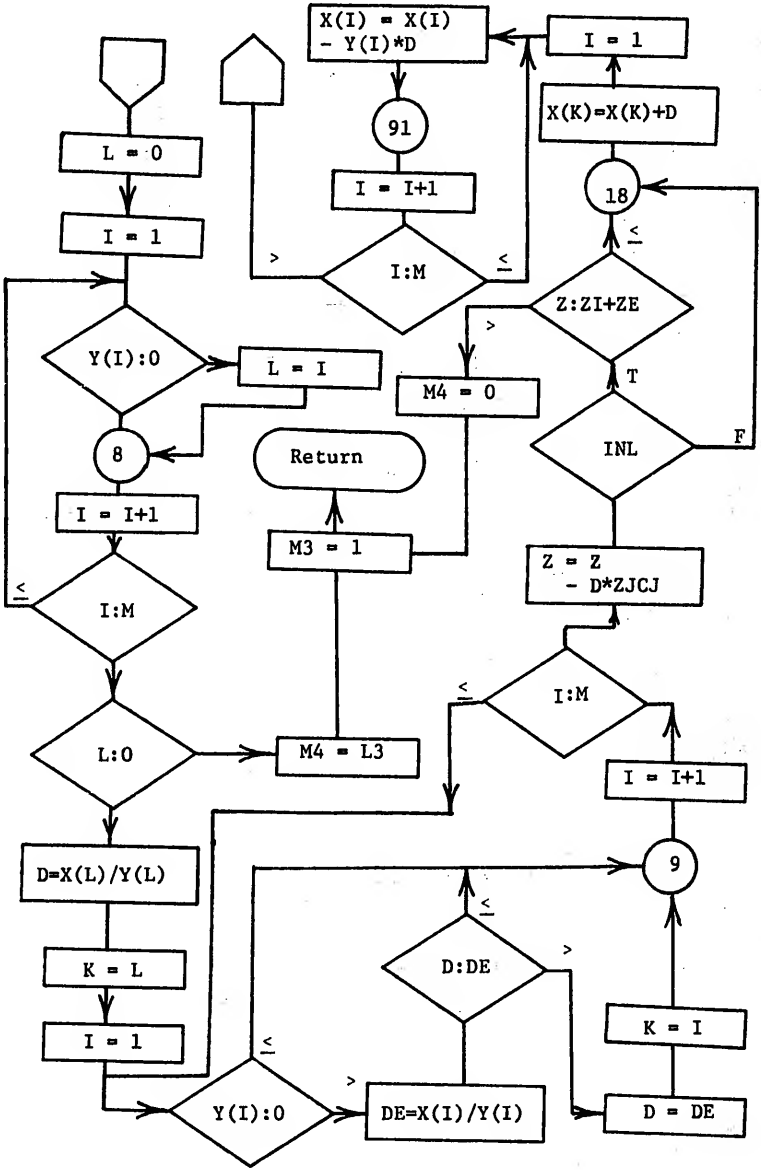


Figure 18. (Continued).

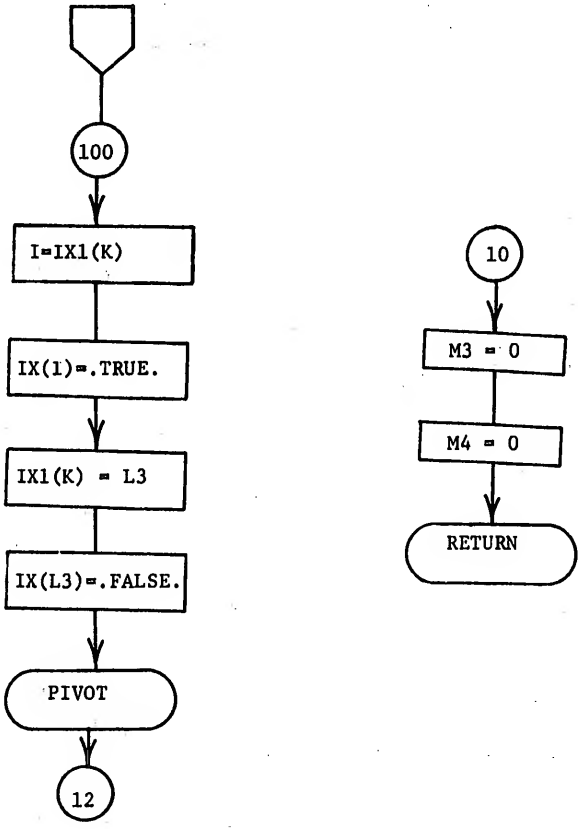


Figure 18. (Continued).

Y for vector L3. Equation (24) gives a formula for Y.

$$a_{L3} = BY_{L3} \tag{24}$$

Iteration loop 7 calculates Y using Equation (83) since we have the inverse matrix, BI.

$$BIa_{L3} = BIBY_{L3} \tag{83}$$

or

$$BIa_{L3} = Y_{L3}$$

Iteration loop 7 finds the nonzero components of a_{L3} by using the compact matrix AA. Iteration loop 7 multiplies rows of BI times the column a_{L3} in the same manner that iteration loop 17 multiplies the vector BI times the columns a_{I1}.

Iteration Loop 8

Iteration loop 8 finds a positive component of the vector Y. If there is no positive component then our problem has an unbounded solution.[15] Control is transferred to the calling program with M3 = 1. M3 = 1 indicates the solution is unbounded. The location of the vector which caused the unbounded solution is given by M4.

Iteration Loop 9

Iteration loop 9 finds the variable K which is to be taken out of the basis. Each basis variable, I, has a value of X(I) in the current solution to the linear programming problem. K is found by Equation (84).

$$\frac{X(K)}{Y(K)} \leq \frac{X(I)}{Y(I)} \qquad \text{for all I such that } X(I) \geq 0 \tag{84}$$

Iteration loop 9 finds K by Equation (84) which assures that the next solution will be feasible.[15] D is defined as X(K)/Y(K) and the new value of the objective function, Z, is given by Equation (85).

$$Z_{new} = Z_{old} - D \times ZJCJ \tag{85}$$

INC, if true, indicates there is an incumbent solution (Chapters III and IV) with objective function ZI. ZE is an error estimate for Z and if Z_{new} is larger than ZI + ZE control is transferred to the calling program since the solution to the current linear program will not be better (have a smaller objective function) than the incumbent solution.

Iteration Loop 91

Iteration loop 91 calculates new values of X by Equations (86) and (87)

$$X(I)_{new} = X(I)_{old} - Y(I)*D \qquad I \neq K \tag{86}$$

$$\begin{aligned} X(K)_{new} &= X(K)_{old}/Y(K) \\ &= D \\ &= X(K) + D - X(K) \\ &= X(K) + D - Y(K) \times D \end{aligned} \tag{87}$$

Control Point 100

IX1 is a vector which indicates which variables are in the basis. When control of the program is transferred to 100, variable IX1(K) is to be taken out of the basis and variable L3 is to be put in the basis in the new solution. We need to set IX(I) = .TRUE. for the variable I to be taken out of the basis and s t IX(L3) = .FALSE. IX1(K) is given its new value of L3.

Control Point 10

Transfer of control to 10 indicates the subprogram has solved its linear programming problem and control is then transferred to the calling program.

Subroutine DEMINF

In the sample problem of Chapter III we first encountered the phenomenon where one decision could imply another. In Chapters IV and V we found other instances where a set of decisions could imply another set of decisions. DEMINF finds relations between decisions by means of data stored in a tensor KB.

Iteration Loop 1

DEMINF is called when a new decision is made by the program FATHOM. FATHOM uses the Balas enumeration method to solve scheduling problems. A parameter, N1, is fed to DEMINF from FATHOM. If a set of N1 decisions have been made in the past then DEMINF can reduce the amount of enumeration necessary in the future.

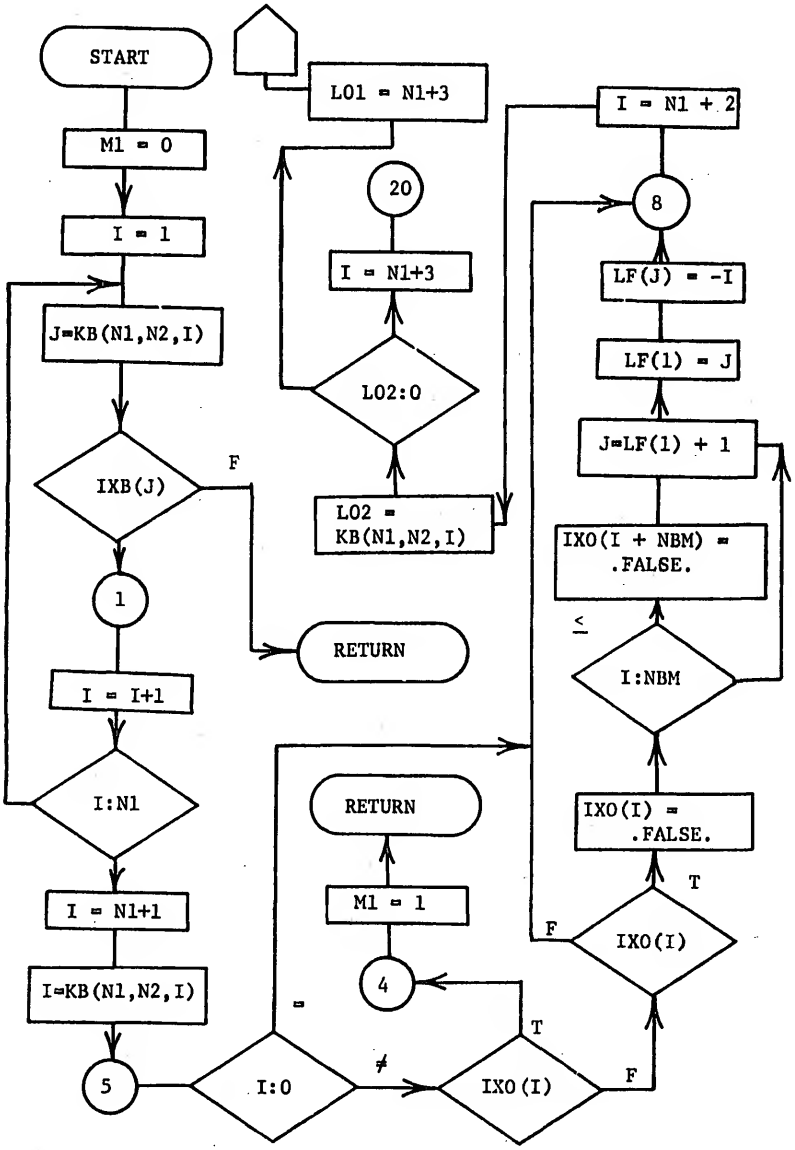


Figure 19. Flow Chart for Subroutine DEMINF.

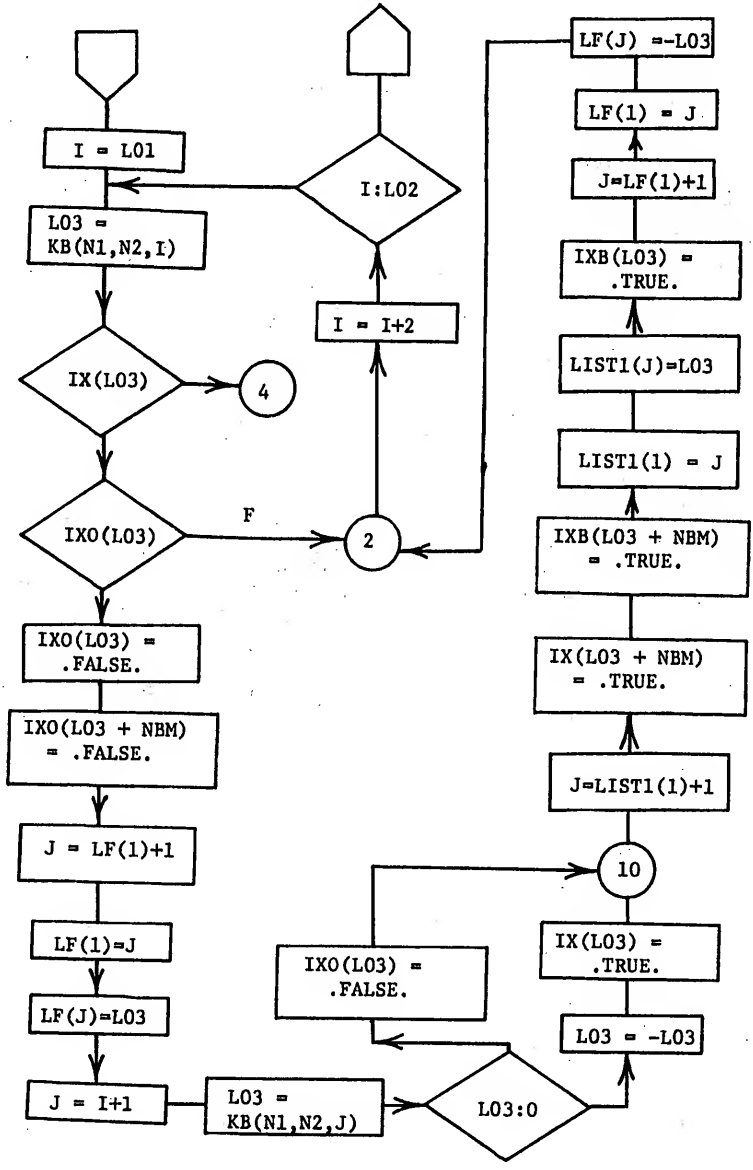


Figure 19. (Continued).

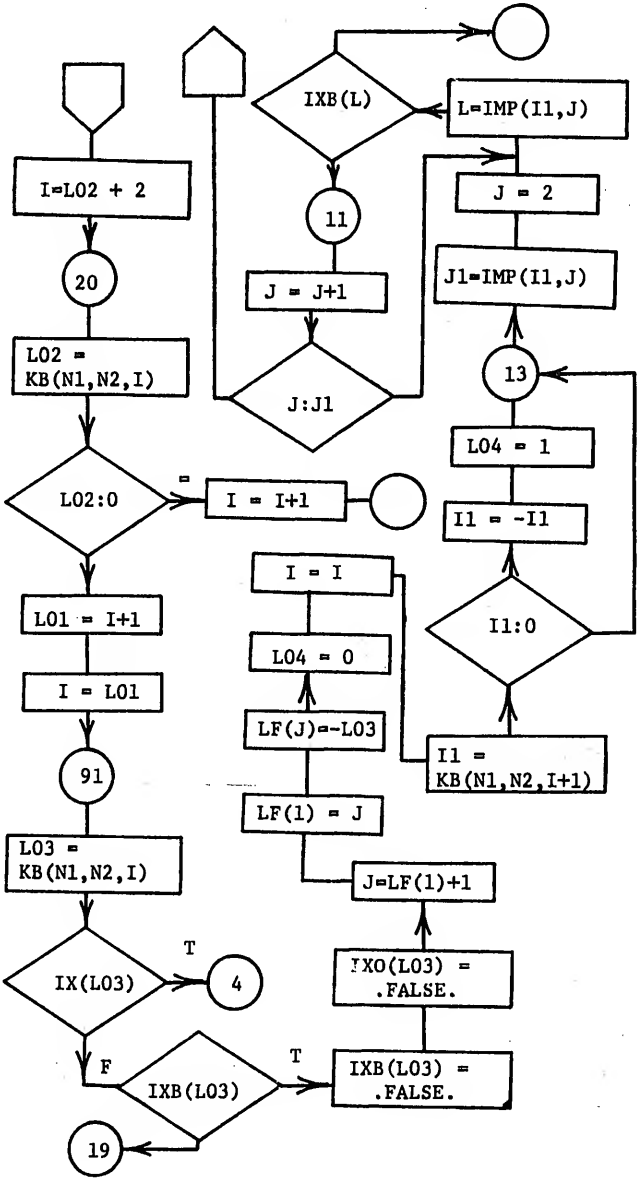


Figure 19. (Continued).

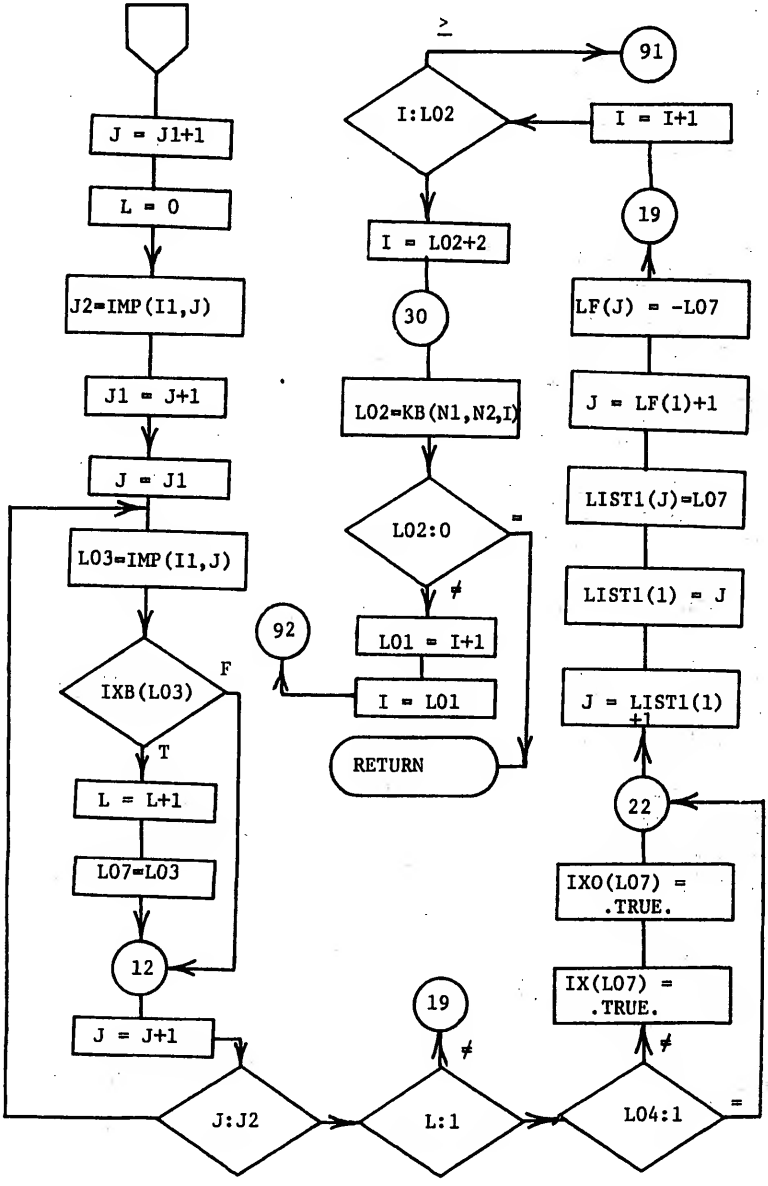


Figure 19. (Continued).

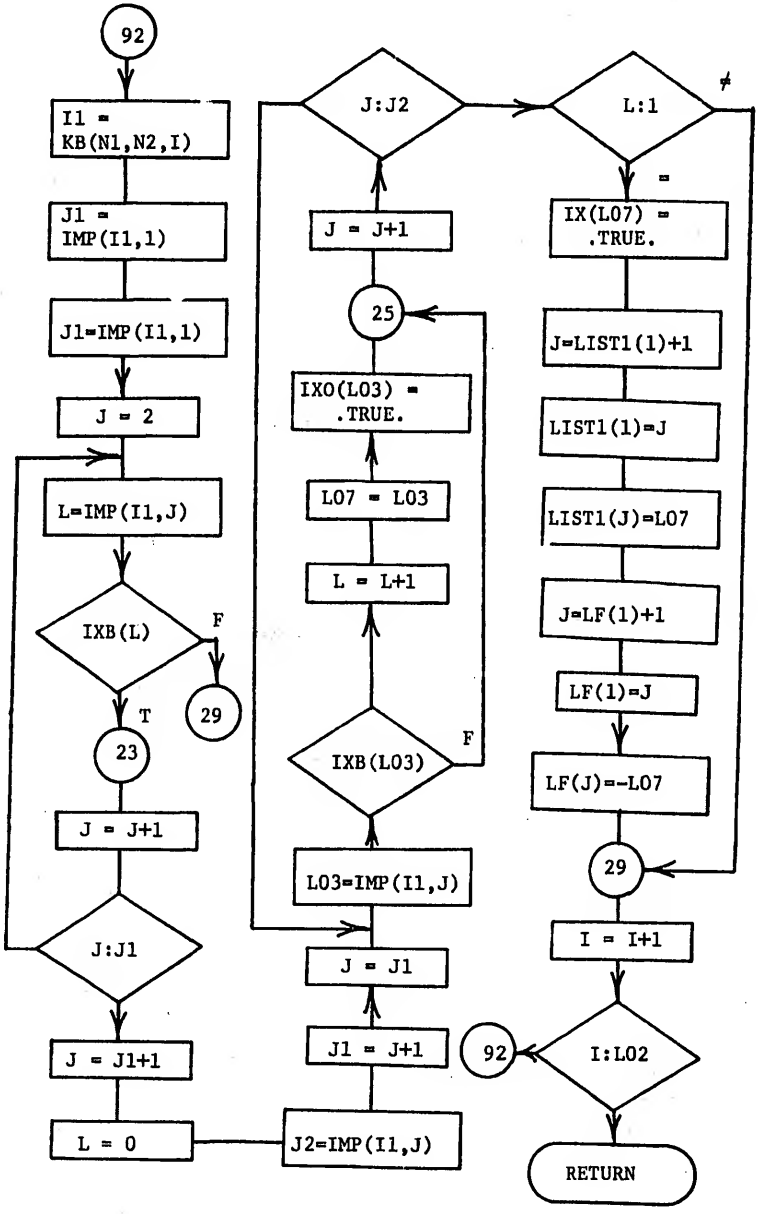


Figure 19. (Continued).

As in Chapter III and later, decisions are made in the dual by either constraining a variable to zero or allowing the variable to take on any positive value. In the description of subroutine LINEAR we saw how IX(I) = .TRUE. indicates variable I is not constrained to zero. If variable I is free this means that constraint I of the primal problem is satisfied.

The vector IX cannot be used to tell which constraints of the original primal problem are not satisfied. In Chapter V (Equation (40) and following) we saw that $x_{ij} \geq 4$ implied $x_{ij} \geq 2$. In the dual the variable corresponding to $x_{ij} \geq 4$ is freed and the variable corresponding to $x_{ij} \geq 2$ is constrained to zero (IX(I) = .FALSE). The variable corresponding to $x_{ij} \geq 2$ adds nothing to the problem since the original constraint is redundant. Even though IX(I) = .FALSE. for the variable in the dual, the original constraint is satisfied.

The programs indicate that a constraint I is not satisfied in the primal by setting IXB(I) = .FALSE. Iteration loop 1 merely finds if a set of N1 constraints are satisfied in the primal. If any member of the set is not satisfied the DEMINF returns control to the calling program, otherwise DEMINF proceeds to reduce the size of the enumeration problem since some decisions are implied by the N1 decisions made in the past.

Control Point 4

DEMINF is usually called when an either/or variable is freed. When one member of an either/or set is freed then the other member must be constrained to zero.

When an either/or variable is freed during some stage of enumeration then the variable, I, which has to be constrained to zero is located by I = KB(N1,N2,N1+1). If variable I has already been freed for some reason then there is some sort of error and control is returned to the calling program. The error is indicated to the calling program by setting a parameter M1 equal to one.

Flow of Control from Point 5 to Point 8

If I = KB(N1,N2,N1+1) = 0 at control point 5 then control is transferred to control point 8. If I is nonzero then I is an either/or variable which must be constrained to zero. IX(I) = .TRUE. would indicate variable I was freed earlier and M1=1 would indicate to the calling program that there is an error.

If a variable I = KB(N1,N2,N1+1) is to be constrained to zero for the current branch of a tree we indicate this by setting IXO(I) = .FALSE.

In Chapter V (Equations (4) through (48)) we saw how variables had to be scheduled if they were made. Variables one through NBM represent decisions to produce batches and the next NBM variables represent the corresponding decisions to schedule them. If I is a variable representing a production decision ($I \leq NBM$) we must make the production decision (IXO(I) = .FALSE.) and the corresponding scheduling decision (IXO(I + NBM) = .FALSE.) simultaneously.

In Figures (7) through (9) of Chapter IV we saw how a list of numbers could indicate the state of the Balas enumeration procedure. The list of numbers is maintained in a vector LF. LF(1) locates the

last number in the list. The number -I is added to LF to indicate a decision has been made about variable I and the opposite decision has been eliminated from consideration.

Control Point 8

The reader should have noted that DEMINF uses information from the list KB(N1,N2,1), KB(N1,N2,2), KB(N1,N2,3),.... There are five types of information in the list. The first type of information is handled by iteration loop 1. The second type of information is handled from iteration loop 1 to control point 8.

The third type of information is handled by the operations between control point 8 and control point 20. The third type of information reduces the number of decisions which have to be made about how many batches should be produced. One may not be able to reduce the amount of work on these types of decisions. If KB(N1,N2,N1+2) = 0 then DEMINF will not be concerned with variables which express decisions about the number of batches to be produced and control is transferred to control point 20.

Iteration Loop 2

Iteration loop 2 is concerned with pairs of either/or variables which represent decisions about how many batches should be produced. Iteration loop 2 reads pairs of numbers from the tensor KB.

The first number, L03, from each pair of numbers represents a variable that is constrained to zero. If variable L03 is free (IX(L03) = .TRUE.), there is an error and control is transferred to control point 4.

If IX0(L03) is false, variable L03 has already been constrained to zero and we look at the next pair of variables in iteration loop 2. Otherwise we set IX0(L03) and IX0(L03 + NBM) false. -L03 is added to the list LF since the enumeration scheme need never look at freeing variable L03 on this branch of the tree.

The second variable of each either/or set could be set free. This may not be necessary since the corresponding constraint might be redundant. For example, if the decision is made that $x_{ij} \geq 4$ in the primal then the dual variable corresponding to $x_{ij} \geq 3$ is redundant and can be constrained to zero. Redundancy is indicated by the sign of the number stored in KB.

Even though we might not need to use the constraint $x_{ij} \geq 3$ we still need to schedule the third batch of i in j so the proper members of IX and IXB are set true.

The location of the second variable of each either/or set is added to LF as a negative number just as the location of the first variable was added to LF.

The location of the second variable of the either/or set has to be added to the vector LIST 1. LIST 1 is used by FATHOM to reduce enumeration still further. An example of this is that $x_{ij} \geq 4$ will imply $x_{ij} \geq 3$. The variable corresponding to $x_{ij} \geq 3$ will be added to LIST 1. FATHOM will use this number to call DEMINF later and find that $x_{ij} \geq 2$ is implied by $x_{ij} \geq 3$.

Control Point 20

Between control point 8 and control point 20 subprogram DEMINF

diminishes the enumeration by making decisions on how many batches should be produced. Between control point 20 and control point 30 the subprogram makes decisions about how batches should be scheduled.

Again the storage location L02 finds decisions which are implied by other decisions. If L02 is zero then no decisions can be made about scheduling batches and control is transferred to control point 30.

Iteration Loop 19

Iteration loop 19 starts with control point 91. After control point 91 we store a number in storage location L03.

L03 is a variable which is to be constrained to zero as a consequence of the decisions the subprogram looked at in loop 1. IX(L03) = .TRUE. would indicate variable L03 is currently not constrained to zero and control should be transferred to point 4. If variable L03 is already constrained to zero (IXB(L03) = .FALSE.), we look for other variables in loop 19.

Normally IX(L03) and IXB(L03) are set false to indicate variable L03 is constrained to zero. -L03 is added to the list LF to indicate one need not consider variable L03 having any value but zero on this stage of the enumeration procedure.

Control Point 13

Variable L03 is a member of a set of either/or variables. Variable L03 will be constrained to zero and variable L04 indicates whether the other variables of the either/or set are redundant.

For example if the decision is made to schedule some batch E before the first of many batches of F, subroutine DEMINF will be called. The variable corresponding to scheduling the second batch of F before batch E will be constrained to zero by DEMINF. The constraint corresponding to scheduling the second batch of F after batch E is redundant so the corresponding variable in the dual can be constrained to zero. Redundancy is indicated by the sign of a number in the tensor KB. L04 is set to 1 if there is redundancy.

Sometimes there is no redundancy. This is true for the first sample problem of Chapter V. In that sample problem, reactors created feed for a separator. If the decision is made not to produce a batch in a reactor, subroutine DEMINF is called. The batch cannot be used to feed the separator so the variable, L03, which corresponds to using the batch to feed the separator, is constrained to zero. The whole set of either/or variables corresponding to feeding the separator cannot be constrained to zero. One of the batches from the reactor has to be used to feed a batch in the separator.

Iteration Loop 11

The parameter I1 is feed to iteration loop 11. After control point 91 we found a variable L03 which was to be constrained to zero. L03 is a member of a set of either/or variables which represent scheduling decisions. One cannot make a decision about scheduling batches unless the decisions are made to make the batches.

Iteration loop 11 uses parameter I1 and the matrix IMP to find decisions about what batches have been made. If the decisions

have been made to produce all of the batches (IXB(L) = .TRUE.) of the either/or set to which L03 belongs then a decision can be made to free a member of the either/or set.

Iteration Loop 12

In iteration loop 11 we found that a number of batches were to be made. After control point 91, a decision was made to constrain one member of an either/or set of scheduling variables to zero. One should remember that we are solving the dual of a scheduling problem. A feasible schedule has to satisfy one constraint from a set of either/or constraints. After control point 91 we decided that one of the constraints would not be satisfied.

Iteration loop 12 finds out how many decisions can still be made about a set of either/or constraints. IXB(L03) = .FALSE. indicates constraint L03 of the original scheduling problem will not be satisfied. If IXB(L07) is .TRUE. for only one member, L07, of the either/or set then constraint L07 must be satisfied.

Control Point 22

At control point 22 we know constraint L07 of the original scheduling is satisfied. If L04 = 1 from control point 13 then variable L07 in the dual can be constrained to zero since scheduling constraint L07 is redundant. If L04 = 0 then dual variable L07 is freed by setting IX(L07) and IXO(L07) = .TRUE.

At any rate scheduling constraint L07 is satisfied and we alter the vectors LIST 1 and LF as we did after control point 10.

Control Point 30

After control point 30 a number LO2 is calculated. If LO2 is zero, control is returned to the calling program. If LO2 is not zero then LO2 helps locate some scheduling decisions.

Whenever a decision is made to produce a batch then DEMINF will be called and after control point 30 LO2 will locate the decisions which are concerned with scheduling the batch.

Iteration Loop 29

Iteration loop 29 starts with control point 92. The subprogram gets to iteration loop 29 because a decision has been made to produce a batch. Iteration loop 29 finds out if some new scheduling decisions can be made as a consequence of the decision to produce a batch.

Iteration Loop 23

Iteration loop 23 performs the same function as iteration loop 11. In both iteration loops the program finds if decisions have been made to produce several batches which will have to be scheduled. A program will transfer control to iteration loop 23 because a decision was previously made to produce one of the batches.

Iteration Loop 25

Iteration loop 25 is the same as iteration loop 12 except for one difference. Iteration loop 25 adds the feature that IXO(LO3) = .TRUE. if IXB(LO3) = .TRUE. IXO(LO3) = .TRUE. says that variable LO3 no longer has to be constrained to zero. Variable LO3 was previously

constrained to zero. Variable L03 was previously constrained to zero because it represented a scheduling decision and decisions had not been made about producing the batches scheduled.

The steps after iteration loop 25 are the same as the steps after iteration loop 12 except there is no provision for redundant variables after loop 25.

Subroutine BACKTR

Subroutine BACKTR performs backtracking for the Balas enumeration procedure of Chapter IV. A list of numbers, LF, is used to show the current state of the enumeration procedure and backtracking is performed as explained in Chapter IV.

Iteration Loop 5

A variable I which is free, but in the basis, is characterized by IX(I) = .FALSE. Iteration loop 5 sets IX(I) = .TRUE. for every variable in the basis. After iteration loop 5 every variable which is free will be characterized by IX(I) = .TRUE.

Control Point 2

List LF is the same list LF described with subroutine DEMINF. The first member of the list, LF(1), is a pointer which locates the last member of the list. If LF(1) is one then an enumeration problem has been fathomed and subroutine NEXTPA is called.

If a scheduling problem is solved using Balas' enumeration then NEXTPA merely writes a message and computation is complete. If a scheduling problem is solved using branch-bound and Balas enumeration

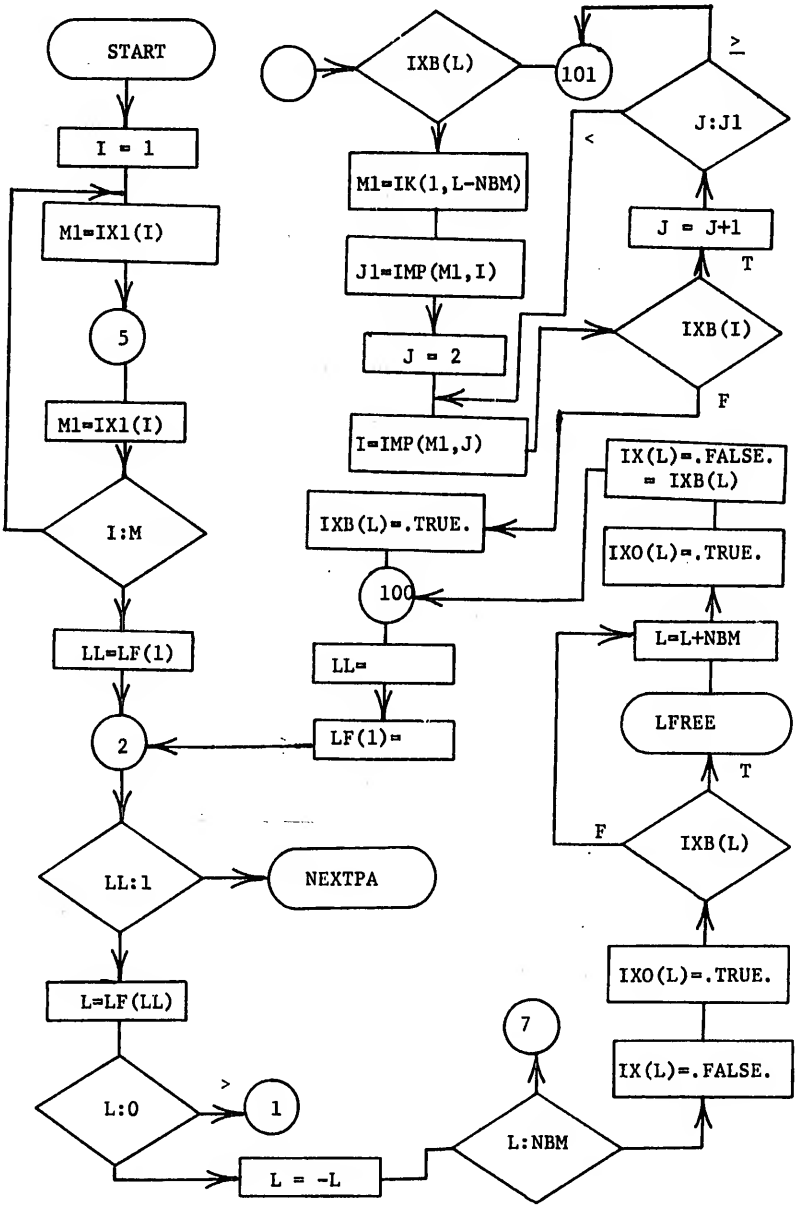


Figure 20. Flow Sheet for Subroutine BACKTR.

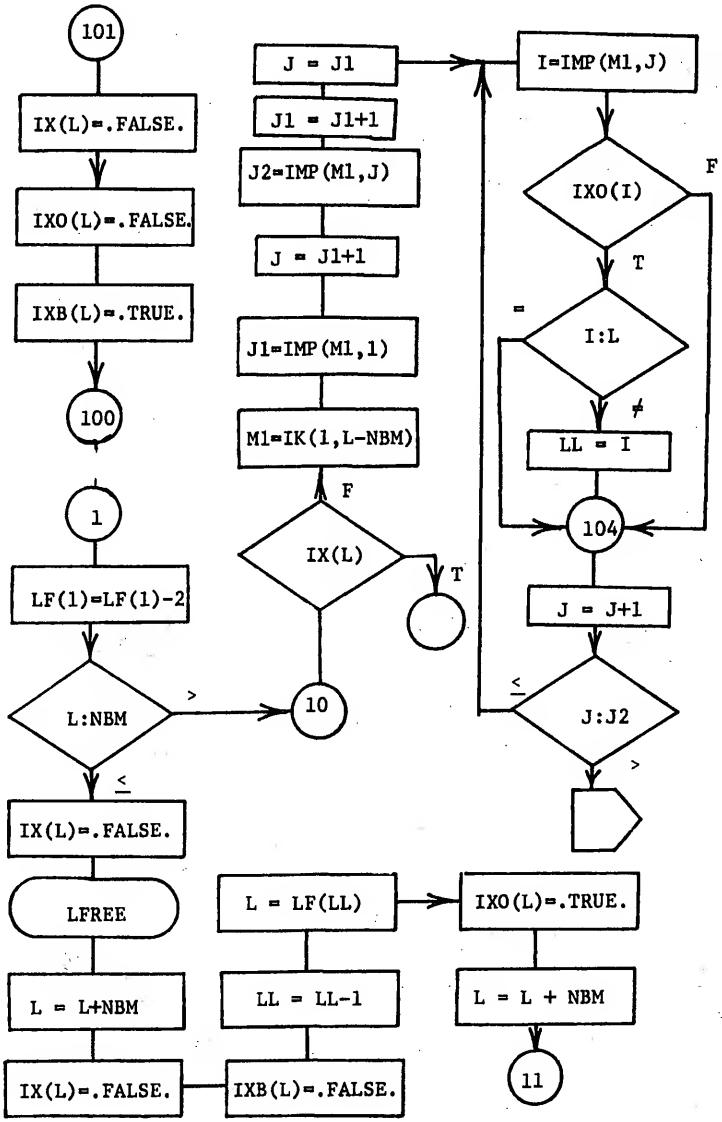


Figure 20. (Continued).

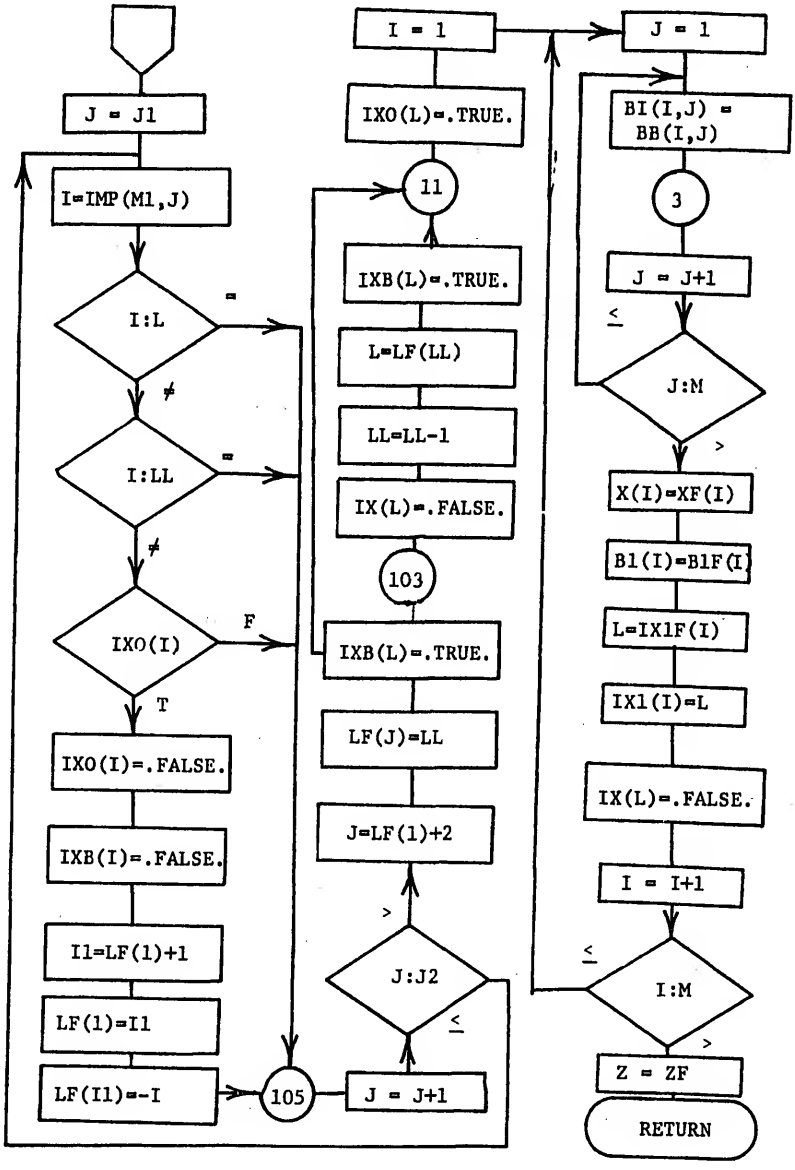


Figure 20. (Continued).

(see last section of Chapter IV), then NEXTPA connects the Balas enumeration programs with the branch-bound programs.

Transfer of Control from Point 2 to 100

The last number in the list LF indicates the last decision taken by the Balas enumeration procedure. This decision was either to free some variable L or to constrain the variable to zero. If the enumeration procedure has already looked at both possibilities then a minus L is stored at the end of list LF. If the program has only enumerated one possibility then a plus L is stored at the end of the list LF.

If the last number in LF is positive control is transferred to control point 1.

If control is not transferred to control point 1, then one has enumerated both possible decisions about variable L. We need to backtrack and enumerate some decisions (partial solutions) that were made earlier. Enumeration of the earlier decisions might involve variable L being constrained to zero or variable L being freed.

If variable L represents a decision about the number of batches produced ($L \leq NBM$) then we set IX(L) = .FALSE. and IXO(L) = .TRUE. IX(L) = .TRUE. would erroneously free variable L during enumeration of all of the earlier decisions. IXO(L) = .FALSE. would erroneously constrain variable L to zero.

IXB(L) = .TRUE. indicates constraint L of the original scheduling problem is satisfied. Subroutine LFREE adjusts scheduling decisions so that earlier partial solutions can be enumerated.

Production decision L and scheduling decision L + NBM are related in the same way that Equations (40) through (43) are related to Equations (44) through (47).

Control Point 100

At control point 100 we shorten the list LF and look at an earlier decision made by the Balas enumeration procedure.

Control Point 7

At control point 7 variable L is a scheduling variable. IXB(L) can only be true if the last decision made by the program was to free variable L. In this case control is transferred to control point 101.

If IXB(L) is .FALSE. the program determines if the decisions have been made to produce the batches scheduled by variable L. If the decision has not been made to produce one of the batches, IXB(L) is set .FALSE. and control is transferred to control point 101.

Control Point 101

At control point 101 the decisions have been made to produce the batches scheduled by variable L. After control point 101 the variables IX, IXO and IXB are set so that variable L can either be constrained to zero or freed.

Control Point 1

If the last member of the list of numbers in LF is positive, the control is transferred to control point one. This last number in the set, L, is a freed variable. The program needs to backtrack and

look at the other possibility, that is, the program needs to look at constraining variable L to zero.

If variable L is a scheduling variable control is transferred to control point 10. If L is not a scheduling variable we need to set the variables IX and IXB to their proper values and call LFREE just as was done after control point 2. The next to the last variable stored in LF was originally constrained to zero and IX0 is set so that it can be freed.

Control Point 11

At control point 11 backtracing is completed. The calling program will be looking at scheduling problems which were not investigated in the past. The calling program uses dual linear programming problems. A linear programming problem moves from one feasible extreme point to another. After control point 11, BACKTR sets the variables, B1, X, BI, and IX1 to their values for a feasible extreme point for any scheduling problem. This feasible extreme point can be found by solving the dual linear programming problem with every either/or variable constrained to zero.

Control Point 10

Control is transferred to point 10 when the last number of list LF is a scheduling variable. If this scheduling variable, L, is currently free control is transferred to point 103 and the program will look at the other choice, that is, constraining variable L to zero.

If control is not transferred to point 103 the program has to

do a good deal of work. The work is concerned with a phenomenon first discussed in Chapter V. In Chapter V an example problem was concerned with which of four possible batches should be used to feed another batch. In this case the dual problem had to free one of four possible variables in order to find a feasible schedule.

Ordinarily the decision to constrain a variable to zero implies that some other variable is freed. This is not true for the example given in the previous paragraph.

The programs are written to handle pairs of either/or variables. When there is a set of three or more variables and only one of them is to be freed the subroutine BACKTR handles this special case with iteration loops 104 and 105.

Iteration Loop 104

Control is transferred to iteration loop 104 because a scheduling variable, L, was constrained to zero and this did not imply that some other variable was free. The computer indicated this by constraining variable L to zero and adding L to the list LF. After control point 1 the pointer LF(1) was shortened by two.

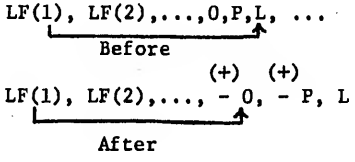


Figure 21. Vector LF Before and After Control Point 1.

Subroutine BACKTR needs to alter the list LF from the way it is shown in Figure 21. Since we are fathoming all possible solutions we need to look at freeing variable L. The other subprograms always assume that there are pairs of either/or variables and subroutine BACKTR has to "fool" these subprograms by adjusting LF.

Iteration loop 104 looks at the set of either/or variables to which L belongs and finds one of the variables, LL, which is free. After iteration loop 104 the pointer for LF is increased by one.

$$\underset{\uparrow}{LF(1),\ LF(2),\ldots,\ \overset{(+)}{-0},\ \overset{(+)}{-P},\ L}$$

Figure 22. Vector LF After Iteration Loop 104.

Iteration Loop 105

Iteration loop 105 also looks at the either/or set to which variable L belongs and constrains any of these variables, Il, which are currently free to zero. The negative of each of these variables is also added to LF(see Figure 23).

$$\underset{\uparrow}{LF(1),\ LF(2),\ldots,\ \overset{(+)}{-0},\ \overset{(+)}{-P},\ -Il_1,\ -Il_2,\ldots,-Il_n}$$

Figure 23. LF After Iteration Loop 105.

Before control is transferred to control point 11, LF is altered to appear as in Figure 24.

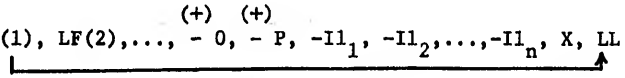


Figure 24. LF After Control Point 11.

List LF as in Figure 24 will "fool" the other programs into thinking that variable LL is currently free and should next be constrained to zero. When variable LL is constrained to zero, the other programs will determine that variable L is the only member of its either/or set which is not constrained to zero. Variable L will be freed as is desired and backtracking with list LF as in Figure 25 will be done correctly.

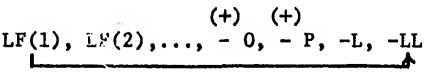


Figure 25. LF After Variable LL Constrained to Zero and Variable L Freed.

Listings

Computer listings follow for the other subprograms. Subprogram FATHOM uses the subprograms described earlier to perform the Balas enumeration developed in Chaptera III, IV and V.

Subprogram BRANCH is essentially the same as subprogram FATHOM except the subprogram uses storage and retrieval of partial solutions instead of fathoming and branching. Subprogram BRANCH stores partial solutions with subprogram STORE.

Variables for partial solutions stored for BRANCH are named by adding BB. For example, the vectors IX1 for stored partial solutions are stored in the matrix IX1BB, the vectors IX are stored in matrix IXBB, etc.

Subprogram BRANCH and the following subprograms are only written for pure scheduling problems. For ease of writing these subprograms use the matrices A and B instead of AA and BI.

The matrix inversion procedure of Appendix A is performed by subprogram UINV. The listings follow.

```
      SUBROUTINE FATHOM(M,IN,NEO)
1     OM=0
      CALL LINEAR(M,IN,M3,M4)
785   IF(M3.EQ.0) GO TO 6
      CALL BACKTR(M,IN,NEO)
      L=LF(1)+2
      L=LF(L)
      M3=3
      M2=1
      GO TO 5
6     L=0
      CDMAX=0.01
      DO 3 I=1,NEO
      IF(IX(I)) GO TO 3
      IF(.NOT.IXO(I)) GO TO 3
      CALL COST(I,CD,M,M1,M2,N1,D)
      IF(M1.EQ.1) GO TO 3
      IF(M2.EQ.0) GO TO 2
      L=I
      GO TO 5
2     IF(CD.LE.CDMAX) GO TO 3
      L=I
      CDMAX=CD
3     CONTINUE
C  IMP-A MATRIX WHICH LOCATES SETS OF VARIABLES WHICH ARE RELATED TO EACH OTHER.
C  IMP IS A MIMP X NIMP MATRIX.
      IF(MIMP.EQ.0) GO TO 191
C  IMP(I,J) J=1,2,...   A VECTOR WHICH SHOWS THE RELATIONSHIP BETWEEN TWO SETS
C             I FIXED   OF VARIABLES.  THE VARIABLES IN ONE SET ARE ALL CON-
C    STRAINED TO ZERO UNLESS ALL OF THE MEMBERS OF ANOTHER SET OF VARIABLES ARE
C  FREE.  LET L=IMP(I,1).  WE LOOK AT THE VARIABLES FOUND IN THE LIST IMP(I,2)
C  THROUGH IMP(I,L).  IF THIS FIRST SET OF VARIABLES ARE FREE ONE MEMBER OF
C  ANOTHER SET OF VARIABLES WILL EVENTUALLY BE FREE.  LET M=IMP(I,L+1).  THE
C  SECOND SET OF VARIABLES ARE FOUND IN THE LIST IMP(I,L+2) THROUGH IMP(I,M).
```

```
C  IXB-THERE ARE TWO TYPES OF EITHER /OR VARIABLES.  FOR OUR SPECIFIC PROBLEM
C  WE HAVE EITHER/OR VARIABLES WHICH DETERMINE THE NUMBER OF BATCHES TO BE MADE
C  AND EITHER/OR VARIABLES WHICH DETERMINE SCHEDULING.  WE DO NOT USE THE SCHEDU
C  LING VARIABLES TO SCHEDULE THE PRODUCTION OF A BATCH UNLESS THE BATCH IS TO
C  BE MADE.  ONCE WE HAVE MADE DECISIONS ON SOME BATCHES-TO-BE MADE VARIABLES WE
C  CAN MAKE DECISIONS ON SOME SCHEDULING VARIABLES..    LET I BE A BATCH-TO-BE-
C  MADE VARIABLE.  IF IXB(I)=TRUE THE BATCH-TO-BE- MADE VARIABLE IS FREE OR
C  IMPLIED FREE BY ANOTHER VARIABLE.
C  LET L BE A SCHEDULING VARIABLE.  THE VARIABLE CAN BE FREED ONLY AFTER A SET,S
C  OF SCHEDULING VARIABLES ARE FREE.  WE CAN CONSTRAIN VARIABLE L TO BE ZERO ON
C  THIS BRANCH OF THE TREE BEFORE WE HAVE MADE DECISIONS ON THE SET S BY
C  SETTING IXB(L)=FALSE.
      DO 190 I=1,MIMP
      I1=0
      J1=IMP(I,1)
      DO 170 J=2,J1
      I2=IMP(I,J)
      IF(IXB(I2)) GO TO 170
      IF(.NOT.IXO(I2)) GO TO 190
      I3=I2
      I1=I1+1
170   CONTINUE
      IF(I1.NE.1) GO TO 190
      J2=IMP(I,J1+1)
      J1=J1+2
      CDMIN=ZUNBND
      DO 175 J=J1,J2
      I1=IMP(I,J)
      IF(.NOT.IXB(I1)) GO TO 175
      CALL COST(I1,CD,M,M1,M2,N1,D)
      IF(M2.EQ.1) GO TO 175
      IF(M1.EQ.1) CD=0.0
      IF(CD.GE.CDMIN) GO TO 175
      CDMIN=CD
```

```
175   CONTINUE
      IF(CDMIN.NE.ZUNBND) GO TO 176
      L=I3
      GO TO 5
176   IF(CDMIN.LE.CDMAX) GO TO 190
      CDMAX=CDMIN
      L=I3
190   CONTINUE
191   CONTINUE
      IF(L.GT.0) GO TO 5
      IF(OM.EQ.1) GO TO 1
      CALL LINEAR(M,IN,M3,M4)
      IF(M3.EQ.0) GO TO 37
      CALL BACKTR(M,IN,NEQ)
      GO TO 38
37    CALL OPTIMA(M)
      CALL BACKTR(M)
38    L=LF(1)+2
      L=LF(L)
      M3=3
      M2=1
5     IF(L.GT.NBM2) GO TO 75
      WRITE(6,787) L
787   FORMAT(' L=',I4)
      IF(L.LE.NBM) GO TO 76
      L=L-NBM
76    I=KB(1,L,2)
      I1=I+NBM
      IXB(I1)=.TRUE.
      IX(I1)=.TRUE.
      GO TO 80
75    IF(L.LE.NSEQ) GO TO 79
      WRITE(6,787) L
      IXQ(L)=.FALSE.
```

```
      IXB(L)=.FALSE.
      I1=IK(1,L-NBM)
      J1=IMP(I1,1)
      J=J1+1
      J2=IMP(I1,J)
      J1=J+1
      M4=0
      DO 78 J=J1,J2
      LO3=IMP(I1,J)
      IF(.NOT.IXB(LO3)) GO TO 78
      M4=M4+1
      I=LO3
78    CONTINUE
      J=LF(1)+1
      LF(1)=J
      IF(M4.EQ.1) GO TO 88
      LF(J)=L
      GO TO 6
88    L4=J
      J=J+1
      LF(1)=J
      LF(J)=-L
      GO TO 90
79    I=KB(1,L-NBM,2)
      WRITE(6,787) L
      IXB(L)=.FALSE.
80    L4=LF(1)+1
      LF(1)=L4
90    LIST1(1)=2
      LIST1(2)=I
      LIST2(1)=1
      IXB(I)=.TRUE.
      IX(I)=.TRUE.
      IXO(I)=.TRUE.
```

```
      DO 81 L3=1,M
      M4=IX1(L3)
81    IX(M4)=.TRUE.
82    IF(LIST1(1).EQ.1) GO TO 86
      J=LIST1(1)
      J=LIST1(J)
      N2=J-NBM
      IF(J.LE.NBM) N2=J
      IF(IK(2,N2).EQ.0) GO TO 85
      L3=IK(2,N2)
      J1=LIST2(1)
      DO 84 J=3,L3,2
      J1=J1+1
      LIST2(J1)=IK(J,N2)
      J1=J1+1
84    LIST2(J1)=IK(J+1,N2)
      LIST2(1)=J1
85    LIST1(1)=LIST1(1)-1
      CALL DEMINF(1,N2,M1)
      IF(M1.EQ.1) GO TO 30
      GO TO 82
86    J=LIST2(1)
      IF(J.EQ.1) GO TO 87
      N2=LIST2(J)
      N1=LIST2(J-1)
      LIST2(1)=J-2
      CALL DEMINF(N1,N2,M1)
      IF(M1.EQ.1) GO TO 30
      GO TO 82
87    CONTINUE
      DO 18 L3=1,M
      M4=IX1(L3)
18    IX(M4)=.FALSE.
      IF(M2.EQ.1) GO TO 8
```

```
      GO TO 9
8     LF(L4)=-I
      IF(M3.EQ.3) GO TO 1
      OM=1
      GO TO 6
9     IF(INC) GO TO 20
44    LF(L4)=L
      L4=L4+1
      LF(L4)=I
      GO TO 1
20    IF((Z+CDMAX).GE.(ZI+ZE)) GO TO 8
      GO TO 44
30    WRITE(6,31)
31    FORMAT(' GOT TO PSEUDO ERROR IN DEMINF')
      WRITE(6,777) IX1,(LF(I),I=1,N1)
      LF(L4)=-I
      CALL BACKTR(M,IN,NEQ)
      L=LF(1)+2
      L=LF(L)
      M3=3
      M2=1
      GO TO 5
      END
      SUBROUTINE LFREE(L)
C  HERE VARIABLE L FREE AND DON'T WANT FREE WHEN BACKTRACK.  NEED TO FIND IF
C  THERE ARE A SET OF VARIABLES WHICH CANNOT BE FREE IF L NOT FREE.
      IXB(L)=.FALSE.
      LO2=KB(1,L,3)
      IF(LO2.EQ.0) GO TO 1
      I=LO2+2
      GO TO 2
1     I=4
2     LO2=KB(1,L,I)
      IF(LO2.EQ.0) GO TO 3
```

```
      I=LO2+2
      GO TO 4
3     I=I+1
4     LO2=KB(1,L,I)
      IF(LO2.EQ.0) RETURN
      LO1=I+1
      DO 6 I=LO1,LO2
      I1=KB(1,L,I)
      J1=IMP(I1,1)
      J=J1+1
      J2=IMP(I1,J)
      J1=J+1
      DO 5 J=J1,J2
      LO3=IMP(I1,J)
5     IXO(LO3)=.FALSE.
6     CONTINUE
      RETURN
      END
      SUBROUTINE COST(L,CO,M,M1,M2,N1,D)
      N1=A(1,L)+1
      ZJCJ=C(L)
      DO 1 I=2,N1
      N2=A(I,L)
1     ZJCJ=ZJCJ+B1(N2)*A(I+N1-1,L)
      IF(ZJCJ.LT.0.0) GO TO 2
      M1=1
      RETURN
2     M1=0
      DO 7 I=1,M
      Y(I)=0
      DO 8 K=2,N1
      N2=A(K,L)
8     Y(I)=Y(I)+B(I,N2)*A(K+N1-1,L)
7     CONTINUE
```

```
      N1=0
      DO 3 I=1,M
3     IF(Y(I).GT.0) N1=I
      IF(N1.GT.0) GO TO 4
      M2=1
      RETURN
4     M2=0
      D=X(N1)/Y(N1)
      DO 5 I=1,M
      IF(Y(I).LE.0) GO TO 5
      DE=X(I)/Y(I)
      IF(D.LE.DE) GO TO 5
      D=DE
      N1=I
5     CONTINUE
      CD=-D*ZJCJ
      RETURN
      END
      SUBROUTINE BRANCH(M,IN,NEO)
1     OM=0
      CALL LINEAR(M,IN,M3,M4)
      IF(M3.EQ.1) CALL NEXTPA(M,IN,NEO)
      IF(Z.LT.(ZF+ZB)) GO TO 6
      CALL SWITCH(M,NEO)
      WRITE(6,400) IX1
400   FORMAT(8I4)
      CALL UINV(M,MU)
      IF(MU.EQ.1) GO TO 119
      GO TO 1
6     L=0
      WRITE(6,300) Z,(ZBB(I),I=1,NBB)
300   FORMAT(E15.6,/,(5E15.6))
      WRITE(6,301) IX1
301   FORMAT(8I4)
```

```
      WRITE(6,302) LBB
302   FORMAT(30I2)
      CDMAX=0.0
      DO 3 I=1,NEO
      IF(IX(I)) GO TO 3
      IF(.NOT.IX0(I)) GO TO 3
      CALL COST(I,CD,M,M1,M2)
      IF(M1.EQ.1) GO TO 3
      IF(M2.EQ.0) GO TO 2
      L=I
      GO TO 5
2     IF(CD.LE.CDMAX) GO TO 3
      L=I
      CDMAX=CD
3     CONTINUE
      IF(L.GT.0) GO TO 5
      IF(OM.EQ.1) GO TO 1
      CALL LINEAR(M,IN,M3,M4)
      IF(M3.EQ.0) CALL OPTIMA(M)
      CALL NEXTPA(M,IN,NEO)
5     I=KB(1,L,3)
      IF(M2.EQ.1) GO TO 38
      IF(.NOT.INC) GO TO 37
      IF((Z+CDMAX).LT.(ZI+ZE)) GO TO 37
      M2=1
      GO TO 38
37    CALL STORE(I,L,CDMAX,M,IN,NEO)
38    IX(I)=.TRUE.
      I1=IK(1,I)
      DO 17 L3=1,M
      M4=IX1(L3)
17    IX(M4)=.TRUE.
      DO 7 L3=2,I1
      N1=IK(L3,I)
```

```
      N2=L3+1
      N2=IK(N2,I)
      CALL DEMIN(N1,N2,M1)
      IF(M1.EQ.1) GO TO 30
7     CONTINUE
      DO 18 L3=1,M
      M4=IX1(L3)
18    IX(M4)=.FALSE.
      IF(M2.EQ.0) GO TO 1
      OM=1
      GO TO 6
119   WRITE(6,120)
120   FORMAT(' THROWN OUT OF UINV')
30    WRITE(6,1000)
1000  FORMAT( 'THERE IS AN ERROR IN BRANCH')
31    STOP
      END
/*
      SUBROUTINE STORE(I,L,CDMAX,M,IN,NEO)
C  IN THIS SUBROUTINE WE WANT TO STORE A LOWER BOUND ON A PARTIAL SOLUTION
C  AND A METHOD OF FINDING THE PARTIAL SOLUTION IF WE NEED TO LOOK AT IT
C  LATER.    IN THE PREVIOUS PARTIAL SOLUTION VARIABLES I AND L WERE NOT
C  CONSIDERED AS CANDIDATES TO ENTER THE BASIS.  WE HAVE A BOUND ON THE
C  PARTIAL SOLUTIONOF THE PREVIOUS CONDITIONS PLUS VARIABLE L BEING FREED.
C NBB-THE NUMBER OF PARTIAL SOLUTIONS STORED SO FAR.
C ZBB-LOWER BOUNDS FOR THE OBJECTIVE FUNCTIONS ARE STORED IN ZBB.
C ZBB(1) IS THE SMALLEST LOWER BOUND FOR ANY OF THE PARTIAL SOLUTIONS
C FOUND SO FAR.  THE LOWER BOUNDS ARE STORED IN NUMERICAL ORDER
C ZBB(NBB) BEING THE LARGEST.
C      WE NEED TO STORE THE VACTORS IX,IXO, AND IX1 FROM EVERY PARTIAL
C  SOLUTION.  THEY ARE STORED IN THE MATRICES IXBB,IXOBB,AND IX1BB.
C LBB-A LIST FOR CORRELATING THE LOCATION OF A STORED LOWER BOUND WITH
C THE OTHER INFORMATION KEPT ABOUT A PARTIAL SOLUTION.  SUPPOSE A
C PARTIAL SOLUTIO N HAS THE THIRD SMALLEST LOWER BOUND.  THE LOWER BOUND
```

```
C IS STORED IN ZBB(3).  THE INFORMATION IN THE VECTOR IX IS STORED IN
C IXBB(1,LBB(3)),IXBB(2,LBB(3)),IXBB(3,LBB(3)...  THE INFORMATION IN THE
C VECTOR S IX0 AND IX1 IS STORED IN IX0BB(I,LBB(3)) AND IX1BB(I,LBB(3)
C I=1,2,... RESPECTIVELY.
C LBB(I)-FOR I LARGER THAN NBB GIVES UNUSED STORAGE LOCATIONS IN IXBB,
C IX0BB, AND IX1BB.
      ZNEW=Z+CDMAX
C ZNEW IS THE LOWER BOUND OF THE NEW PARTIAL SOLUTION.
      NBB=NBB+1
C WE ARE GOING TO STORE STORE ANOTHER PARTIAL SOLUTION.
      M4=LBB(NBB)
C THE STORAGE LOCATIONS USED WILL BE IXBB(I,M4),IX0BB(I,M4),IX1BB(I,M4)
C I=1,2,...
      ZBB(NBB)=ZNEW
      DO 1 J=1,NBB
      IF(ZNEW.LE.ZBB(J)) GO TO 2
1     CONTINUE
C IN LOOP 1 WE FIND THE FIRST LOWER BOUND STORED THAT IS LARGER OR EQUAL
C TO ZNEW.  THE FIRST LOWER BOUND FOUND LARGER THAN ZNEW IS ZVV(J)
C TO ZNEW.  THE FIRST LOWER BOUND FOUND LARGER THAN ZNEW IS ZBB(J).
      WRITE(6,3)
3     FORMAT( 'THERE IS AN ERROR IN STORE')
      STOP
C FROM STATEMENTS 2 TO 5 WE STORE DATA ON OUR PARTIAL SOLUTION.
2     LBL(M4)=L
      DO 4 J1=1,NEQ
      IXBB(J1,M4)=IX(J1)
4     IX0BB(J1,M4)=IX0(J1)
      IXBB(L,M4)=.TRUE.
      LBL(M4)=L
      DO 5 J1=1,M
5     IX1BB(J1,M4)=IX1(J1)
C  WE ARE GOING TO STORE THE LOWER BOUND ZNEW IN LOCATION ZBB(J).  THE
C LOWER BOUNDS HIGHER THAN ZNEW WILL BE LOCATED IN A DIFERENT POSITION
```

```
C IN ZBB THAN THEY WERE LOCATED BEFORE..
      DO 6 J1=J,NBB
      ZINT=ZBB(J1)
      MINT=LBB(J1)
      ZBB(J1)=ZNEW
      LBB(J1)=M4
      ZNEW=ZINT
6     M4=MINT
      ZF=ZBB(1)
      IF(NBB.GE.NMAX) CALL TOOBIG(I,L,M,IN,NEO)
      RETURN
      END
      SUBROUTINE DEMIN(N1,N2,M1)
      M1=0
      DO 1 I=1,N1
      J=KB(N1,N2,I)
      IF(IX(J)) GO TO 1
      GO TO 3
1     CONTINUE
      IF(N1.NE.1) GO TO 7
      I=N1+2
      L03=KB(N1,N2,I)
      IF(IX(L03)) GO TO 4
      IX0(L03)=.FALSE.
7     J=N1+1
      L01=J+2
      L02=KB(N1,N2,J)
      IF(L02.EQ.1) GO TO 3
      DO 2 I=L01,L02,2
      L03=KB(N1,N2,I)
      IF(IX(L03)) GO TO 4
      IF(IX0(L03)) GO TO 5
      GO TO 2
5     IX0(L03)=.FALSE.
```

```
      J=I+1
      LO3=KB(N1,N2,J)
6     IXO(LO3)=.FALSE.
2     CONTINUE
      GO TO 3
4     M1=1
3     CONTINUE
      RETURN
      END
      SUBROUTINE SWITCH(M,NEO)
      NBB=NBB+1
C FIRST WE NEED TO STORETHE PRESENT SOLUTION WE ARE WORKING ON.  WE STORE THE
C  INFORMATION ON THE PRESENT SOLUTION IN STORAGE LOCATION M4
      M4=LBB(NBB)
      ZBB(NBB)=Z
C  WE TEMPORARILY STORE THE OBJECTIVE FUNCTION FOR THE CURRENT SOLUTION IN
C  ZBB(NBB).  WE WANT THE OBJECTIVE FUNCTIONS TO BE IN NUMERICAL ORDER,HOWEVER.
C  BEFORE WE CAME TO SWITH
C  BEFORE WE CAME TO SWITCH THE OBJECTIVE FUNCTIONS WERE IN NUMERICAL ORDER.
C  IN LOOP ONE WE FIND OBJECTIVE FUNCTIONS WHICH ARE LESS THAN ZBB(NBB) AND
C  THOSE WHICH ARE GREATER
      DO 1 J=1,NBB
      IF(Z.LE.ZBB(J)) GO TO 2
1     CONTINUE
C  WE WANT TO STORE A ZERO IN LBL(M4) TO SHOW THAT WE ARE SWITCHING FROM A
C  CURRENT SOLUTION AND THE SIZE OF THE PROBLEM CANNOT BE DECREASED BECAUSE
C  OF IT'S STRUCTURE.
2     LBL(M4)=0
C  IN LOOPS THREE AND FOUR WE SIMPLY STORE INFORMATION ON THE CURRENT SOLUTION
C  IN VECTORS M4 OF  THE MATRICES IXBB,IXOBB,AND IX1BB.
      DO 3 J1=1,NEO
      IXBB(J1,M4)=IX(J1)
3     IXOBB(J1,M4)=IXO(J1)
      DO 4 J1=1,M
```

```
4       IX1BB(J1,M4)=IX1(J1)
C   IN LOOP 5 WE ARE GOING TO ARRANGE THE OBJECTIVE FUNCTIONS IN NUMERICAL ORDER.
C   BEFORE THE CURRENT SOLUTION THE OBJECTIVE FUNCTIONS WERE ALREADY IN NUMER-
C   ICAL ORDER AND LOOP ONE SHOWED US THAT THE CURRENT OBJECTIVE FUNCTION SHOULD
C   BE STORED IN LOCATION J OF THE VECTOR ZBB.
        DO 5 J1=J,NBB
        ZINT=ZBB(J1)
        MINT=LBB(J1)
        ZBB(J1)=Z
        LBB(J1)=M4
        Z=ZINT
5       M4=MINT
C   WE ARE  GOING TO SWITCH TO A NEW BRANCH OF THE TREE.    IF A VARIABLE,O,
C   WAS NONZERO IN THE LAST SOLUTION WE NOTED THAT IX(O) WAS FALSE FOR THE
C  PURPOSE OF INFORMING THE SUBPROGRAM LINEAR THAT VARIABLE O WAS NONZERO.
C   IN LOOP 10 WE NOTE THAT VARIABLE O WILL PROBABLY BE ZERO IN THE NEXT
C   BRANCH SO WE NO LONGER WANT IX(O) TO BE FALSE.
        DO 10 J1=1,M
        J=IX1(J1)
10      IX(J)=.TRUE.
        M4=LBB(1)
C   WE WANT TO SWITCH OUR CONSIDERATION THE PARTIAL SOLUTION WITH THE BEST
C   OBJECTIVE FUNCTION.  THE INFORMATION ABOUT THIS PARTIAL SOLUTION IS FOUND IN
C   VECTOR M4 OF THE MATRICES IXBB,IXOBB,IX1BB.
        NBB=NBB-1
C   WE WILL ONLY HAVE TO STORE INFORMATION ABOUT THE CURRENT PARTIAL SOLUTION IN
C   ONE LOCATION.
C   IN LOOPS 6 AND 7 WE BRING INFORMATION ON THE NEW PARTIAL SOLUTION  INTO THE
C   THE ACTIVE STORAGE.
        DO 6 J1=1,NEO
        IX(J1)=IXBB(J1,M4)
6       IXO(J1)=IXOBB(J1,M4)
        DO 7 J1=1,M
        J=IX1BB(J1,M4)
```

```
      IX1(J1)=J
7     IX(J)=.FALSE.
      I=LBL(M4)
C   IF I IS NOT ZERO WE CAN DIMINISH THE WORK NEEDED TO BE DONE ON THE NEW
C   PARTIAL SOLUTION BY USING THE SUBROUTINE DEMIN.
      IF(I.EQ.0) GO TO 20
C   THE NEW PARTIAL SOLUTION WAS DISREGARDED EARLIER BECAUSE IT LOOKED THAT WE
C   WOULD NOT WANT TO LOOK AT THE SOLUTIONS WITH VARIABLE I FREE.
      IX(I)=.TRUE.
      I1=IK(1,I)
      DO 17 L3=1,M
      M5-IX1(L3)
17    IX(M5)=.TRUE.
C   NOW WE WILL LOOK AT PARTIAL SOLUTIONS WITH VARIABLE I FREE AND LOOP 19 TRIES
C   TO FIND WAYS TO REDUCE THE SIZE OF THE PROBLEM BY USING DEMIN.
      DO 19 L3=2,I1
      N1=IK(L3,I)
      N2=L3+1
      N2=IK(N2,I)
      CALL DEMIN(N1,N2,M1)
      IF(M1.EQ.1) GO TO 30
19    CONTINUE
      DO 18 L3=1,M
      M5=IX1(L3)
18    IX(M5)=.FALSE.
C   THE NEW PARTIAL SOLUTION WE ARE GOING TO WORK ON PREVIOUSLY HAD IT'S
C   OBJECTIVE FUNCTION STORED IN ZBB(1).  IN LOOP 8 WE MOVE THE LOCATION OF ALL
C   THE OTHER OBJECTIVE FUNCTIONS STORED UP ONE PLACE AND THE VECTOR LBB HAS TO
CC  BE CHANGED ACCORDINGLY.
20    DO 8 J1=1,NBB
      J=J1+1
      LBB(J1)=LBB(J)
8     ZBB(J1)=ZBB(J)
      J=NBB+1
```

```
      LBB(J)=M4
      ZF=ZBB(1)
      RETURN
30    WRITE(6,31)
31    FORMAT(' WE HAD AN ERROR IN DEMIN AS CALLED BY SWITCH')
      STOP
      END
      SUBROUTINE NEXTPA(M,IN,NEO)
      WRITE(6,1)
1     FORMAT(' WE EXITED THROUGH NEXTPA')
C  IF NBB=0 WE HAVE NO PARTIAL SOLUTIONS STORED AND WE ARE FINISHED.
      IF(NBB.EQ.0) GO TO 14
C     IF WE HAVE AN INCUMBENT SOLUTION WE ARE GOING TO TRY TO ELIMINATE SOME OF
C  THE SOLUTIONS STORED FROM CONSIDERATION.
      IF(.NOT.INC) GO TO 17
C  IF THE PARTIAL SOLUTION WITH THE LARGEST OBJECTIVE FUNCTION CANNOT BE ELIMINA
C  TED THEN NONE OF THE PARTIAL SOLUTIONS CAN BE ELIMINATED.
      IF(ZBB(NBB).LT.ZI+ZE) GO TO 17
C  IN LOOP 18 WE FIND THE FIRST PARTIAL SOLUTION STORED WHICH HAS AN OBJECTIVE F
C  FUNCTION WHICH IS DEFINITELY LARGER THAN THE INCUMBENT.
      DO 18 J1=1,NBB
      IF(ZBB(J1).GT.ZI+ZE) GO TO 19
18    CONTINUE
C  IF J1 =0 WE HAVE ELIMINATED ALL PARTIAL SOLUTIONS FROM CONSIDERATION.
19    IF(J1.EQ.1) GO TO 14
      NBB=J1-1
C  WE ARE GOING TO SWITCH OUR CONSIDERATION TO A NEW PARTIAL SOLUTION AND LOOPS
C . 16,6, AND 7 PUT THE NEW INFORMATION INTO THE CURRENT WORKING MEMORY.
17    DO 16 J1=1,M
      J=IX1(J1)
16    IX(J)=.TRUE.
      M4=LBB(1)
      NBB=NBB-1
      Z=ZBB(M4)
```

```
      DO 6 J1=1,NEO
      IX(J1)=IXBB(J1,M4)
6     IXO(J1)=IXOBB(J1,M4)
      DO 7 J1=1,M
      J=IX1BB(J1,M4)
      IX1(J1)=J
7     IX(J)=.FALSE.
C  IF THERE WILL BE NO PARTIAL SOLUTIONS IN STORAGE WE NEED A SYNTHETIC VALUE
C  IN PLACE OF WHERE WE USED TO STORE THE BEST OBJECTIVE FUNCTION OF ALL PAR-
C  TIAL SOLUTIONS.
      IF(NBB.EQ.0) GO TO 9
C  WE ARE GOING TO USE THE PARTIAL SOLUTION WHOSE  OBJECTIVE FUNCTION WAS STORED
C  IN ZBB(1).  WE NO LONGER NEED TO STORE THE INFORMATION ON THIS PARTIAL
C  SOLUTION IN INACTIVE STORAGE AND LOOP 8 UPDATES OUR INACTIVE STORAGE.
      DO 8 J1=1,NBB
      J=J1+1
      LBB(J1)=LBB(J)
8     ZBB(J1)=ZBB(J)
      J=NBB+1
      LBB(J)=M4
      ZF=ZBB(1)
      GO TO 10
9     ZF=1000.
C  UINV GENERATES A NEW BASIS MATRIX FROM IX1
10    CALL UINV(M,MU)
      IF(MU.EQ.1) GO TO 11
      I=LBL(M4)
C  IF I IS NONZERO WE CAN REDUCE THE WORK TO BE DONE ON THE NEW PARTIAL
C  SOLUTION BY USING DEMINF.
      IF(I.EQ.0) GO TO 20
      I1=IK(1,I)
      DO 21 L3=1,M
      M4=IX1(L3)
21    IX(M4)=.TRUE.
```

```
      DO 22 L3=2,I1
      N1=IK(L3,I)
      N2=L3+1
      N2=IK(N2,I)
      CALL DEMIN(N1,N2,M1)
      IF(M1.EQ.1) GO TO 30
22    CONTINUE
      DO 23 L3=1,M
      M4=IX1(L3)
23    IX(M4)=.FALSE.
C  HERE WE HAVE PUT THE INFORMATION ON A NEW PARTIAL SOLUTION IN ACTIVE STORAGE
C  AND WE CAN RETURN TO BRANCH TO WORK ON THIS NEW PARTIAL SOLUTION.
20    CALL BRANCH(M,IN,NEO)
11    WRITE(6,12)
12    FORMAT(' WE HAD ERROR IN UINV AS CALLED BY NEXTPA')
      STOP
14    WRITE(6,15)
15    FORMAT(' ALL PARTIAL SOLUTIONS HAVE BEEN FATHOMED OR ELIMINATED')
      STOP
30    WRITE(6,31)
31    FORMAT(' WE HAD AN ERROR IN DEMIN AS CALLED BY NEXTPA')
      STOP
      END
      SUBROUTINE UINV(M,MU)
      MU=0
C  MU WILL BE STILL EQUAL TO 0 AT END OF PROGRAM IF DATA WAS O.K. OTHERWISE =1
C     A IS A HIGHLY STRUCTURED MATRIX WITH AT MOST TWO NOZERO TERMSPER COLUMN.
CONLY THE LOCATION OF THE NONZERO TERMS AND THEIR VALUES ARE STORED.  A(1,J)
C=THE NUMBER OF NONZERO TERMS IN COLUMN J.  A(2,J)IS THE ROW WHERE THE FIRST
CNONZERO TERM IS LOCATED.  IF A(1,J)=1,THEN A(3,J)IS THE VALUE OF THE NONZERO
CA(4,J)IS THE VALUE OF THE FIRST NONZERO TERM...
C  WE WANT THE INVERSE OF A MATRIX FORMED FROM COLUMNS OF A. INVERSE CALLED B.
C  IX1(1)=FIRST COLUMN OF A USED,IX1(2)=SECOND COLUMN USED,...
C  M IS THE NUMBER OF ROWS IN MATRIX.
```

```
C SET ALL TERMS OF B TO ZERO.
      DO 2 I=1,M
      DO 1 J=1,M
1     B(I,J)=0
2     CONTINUE
      L05(1)=0
      L06(1)=0
      L25(1)=0
      DO 4 J=1,M
      M1=IX1(J)
      IF(A(1,M1).EQ.2) GO TO 7
      M2=A(2,M1)
      L05(1)=L05(1)+1
      L01=L05(1)+1
      L05(L01)=M2
      L06(J)=L01
      GO TO 4
7     L25(1)=L25(1)+1
      L01=L25(1)+1
      L25(L01)=M1
4     CONTINUE
      IF(L05(1).EQ.0) GO TO 120
      DO 100 I=1,M
      L1(1)=0
      LM1(1)=0
      L01=L25(1)+1
      DO 6 J=1,L01
6     L2(J)=L25(J)
      M1=IX1(I)
      IF(A(1,M1).EQ.2) GO TO 50
      M2=A(2,M1)
      IF(A(3,M1).EQ.(-1)) GO TO 9
      B(I,M2)=1
      L1(1)=1
```

```
      L1(2)=M2
      GO TO 10
9     B(I,M2)=-1
      LM1(1)=1
      LM1(2)=M2
10    CONTINUE
      IF(L05(1).EQ.M) GO TO 100
      L0(1)=L05(1)-1
      L01=L06(I)-1
      DO 11 J=2,L01
11    L0(J)=L05(J)
      L01=L06(I)
      L02=L05(1)
      DO 12 J=L01,L02
12    L0(J)=L05(J+1)
33    CONTINUE
      L=L2(1)+1
      DO 44 J=2,L
      L22=A(2,(L2(J)))
      L23=A(3,(L2(J)))
      IF(L1(1).EQ.0) GO TO 43
      L3=L1(1)+1
      DO 38 K=2,L3
      IF(L1(K).EQ.L22) GO TO 35
      IF(L1(K).EQ.L23) GO TO 36
      GO TO 38
35    L1(1)=L1(1)+1
      L01=L1(1)+1
      L1(L01)=L23
      B(I,L23)=1
      GO TO 39
36    L1(1)=L1(1)+1
      L01=L1(1)+1
      L1(L01)=L22
```

```
      B(I,L22)=1
      GO TO 39
38    CONTINUE
40    IF(LO(1).EQ.0) GO TO 44
      L3=LO(1)+1
      DO 41 K=2,L3
      IF(LO(K).NE.L22) GO TO 47
      LO(1)=LO(1)+1
      LO1=LO(1)+1
      LO(LO1)=L23
      GO TO 39
47    IF(LO(K).NE.L23) GO TO 41
      LO(1)=LO(1)+1
      LO1=LO(1)+1
      LO(LO1)=L22
      GO TO 39
41    CONTINUE
      GO TO 44
43    L3=LM1(1)+1
      DO 49 K=2,L3
      IF(LM1(K).EQ.L22) GO TO 45
      IF(LM1(K).EQ.L23) GO TO 46
      GO TO 49
45    LM1(1)=LM1(1)+1
      LO1=LM1(1)+1
      LM1(LO1)=L23
      B(I,L23)=-1
      GO TO 39
46    LM1(1)=LM1(1)+1
      LO1=LM1(1)+1
      LM1(LO1)=L22
      B(I,L22)=-1
      GO TO 39
49    CONTINUE
```

```
      GO TO 40
39    L2(1)=L2(1)-1
      IF(L2(1).LE.0) GO TO 100
      L3=L2(1)+1
      DO 42 K=J,L3
42    L2(K)=L2(K+1)
      GO TO 33
44    CONTINUE
      GO TO 120
50    L01=L05(1)+1
      DO 51 J=1,L01
51    L0(J)=L05(J)
56    L3=L2(1)+1
C   IN THE LOOP ENDING IN 65 WE WILL FIND THOSE COLUMNS OF THE LIST L2 WHICH HAVE
C   A(2,L2(J)) OR A(3,L2(J)) EQUAL TO L0(K) FOR SOME K
      DO 65 J=2,L3
      L22=A(2,L2(J))
      L23=A(3,L2(J))
      IF(L0(1).LE.0) GO TO 120
      L0K=L0(1)+1
      DO 62 K=2,L0K
      IF(L22.EQ.L0(K)) GO TO 57
      IF(L23.EQ.L0(K)) GO TO 59
62    CONTINUE
      GO TO 65
57    IF(L2(J).EQ.M1) GO TO 75
      L0(1)=L0(1)+1
      L01=L0(1)+1
      L0(L01)=L23
      GO TO 63
59    IF(L2(J).EQ.M1) GO TO 75
      L0(1)=L0(1)+1
      L01=L0(1)+1
      L0(L01)=L22
```

```
63      LOK=L2(1)
        L2(1)=L2(1)-1
        DO 64 K=J,LOK
64      L2(K)=L2(K+1)
        GO TO 56
C HERE WE HAVE MADE THE LIST LO LARGER AND THE LIST L2 SMALLER SO WE REPEAT LOOP
65      CONTINUE
        GO TO 120
C EVENTUALLY WE HAVE TO OBTAIN A MEMBER OF LO SUCH THAT LO(I) EQUALS M2 OR M3
75      IF(L22.EQ.LO(K)) GO TO 77
        IF(A(4,M1).EQ.1) GO TO 76
C  HERE B(I,(A(5,M1))) EQUALS ZERO AND WE HAVE TO ADJUST B(I,(A(4,M1))) SO THAT
C THE ITH ROW OF B TIMES THE ITH ROW OF THE MATRIX TO INVERT WILL BE ONE
        LM1(1)=LM1(1)+1
        LO1=LM1(1)+1
        LM1(LO1)=L22
        B(I,L22)=-1
        GO TO 79
76      L1(1)=L1(1)+1
        LO1=L1(1)+1
        L1(LO1)=L22
        B(I,L22)=1
        GO TO 79
77      IF(A(5,M1).EQ.1) GO TO 78
        LM1(1)=LM1(1)+1
        LO1=LM1(1)+1
        LM1(LO1)=L23
        B(I,L23)=-1
        GO TO 79
78      L1(1)=L1(1)+1
        LO1=L1(1)+1
        L1(LO1)=L23
        B(I,L23)=1
79      IF(L2(1).EQ.1) GO TO 100
```

```
      LOK=L2(1)
      L2(1)=L2(1)-1
      DO 80 K=J,LOK
80    L2(K)=L2(K+1)
      GO TO 33
C HERE WE HAVE FINALLY FOUND THAT THE ITH COLUMN OF THE MATRIX TO INVERT
C HAS A NONZERO TERM IN THE PTH AND QTH ROWS BUT THE PTH COLUMN OF THE ITH
C ROW OF B HAS TO BE ZERO.  WE HAVE A LIST OF THOSE COLUMNS OF THE ITH ROW
C OF B WHICH HAVE TO BE ZERO INCLUDING THE PTH AND A LIST OF THOSE COLUMNS
C OF B WHICH HAVE TO BE PLUS OR MINUS ONE
C INCLUDING THE QTH.  AT 33 WE WILL FIND THE REMAINING COLUMNS OF B WHICH ARE
C NONZERO.
100   CONTINUE
      GO TO 140
120   MU=1
140   CONTINUE
      DO 236 I=1,M
      B1(I)=0.0
      X(I)=0.0
      DO 235 K=1,M
      M1=IX1(K)
      X(I)=X(I)+B(I,K)*RS(K)
235   B1(I)=B1(I)-C(M1)*B(K,I)
236   CONTINUE
      Z=0.0
      DO 231 I=1,M
      M1=IX1(I)
231   Z=Z-C(M1)*X(I)
      RETURN
      END
      SUBROUTINE TOOBIG(I,L,M,IN,NEO)
      IX(I)=.TRUE.
      I1=IK(1,I)
      DO 17 L3=1,M
```

```
      M4=IX1(L3)
17    IX(M4)=.TRUE.
      DO 7 L3=2,I1
      N1=IK(L3,I)
      N2=L3+1
      N2=IK(N2,I)
      CALL DEMIN(N1,N2,M1)
      IF(M1.EQ.1) GO TO 30
7     CONTINUE
      DO 18 L3=1,M
      M4=IX1(L3)
18    IX(M4)=.FALSE.
      IF(.NOT.INC) GO TO 29
      IF(ZBB(NBB).LT.ZI+ZE) GO TO 29
      NBB=NBB-1
      CALL BRANCH(M,IN,NEO)
29    DO 1 I=1,M
      DO 2 J=1,M
2     BB(I,J)=B(I,J)
1     CONTINUE
      DO 3 I=1,M
      B1F(I)=B1(I)
      XF(I)=X(I)
3     IX1F(I)=IX1(I)
      ZF=Z
      CALL FATHOM(M,IN,NEO)
30    WRITE(6,31)
31    FORMAT(' WE EXITED THRU DEMIN ERROR WHILE IN SUBROUTINE TOOBIG')
      STOP
      END
```

BIBLIOGRAPHY

1. Dzielinski, B. P. and Gomery, R. E., Management Sci., 11, 1, 1-32 (1964).

2. Urioste, German R., "Optimal Scheduling of Batch Chemical Processes," Master of Science Thesis, University of Florida, December, 1969.

3. Conway, R. W., Maxwell, W. L., and Miller, L. W., "Theory of Scheduling," Addison-Wesley, Reading, Mass. (1967).

4. Balinski, M. L., Management Sci., 12, 3, 253-313 (1965).

5. Muth, J. F. and Thompson, G. L., "Industrial Scheduling," Prentice-Hall, Englewood Cliffs, N.J. (1963).

6. Lasdon, E. S., "Optimization Theory for Large Systems," The Macmillan Company, New York (1970).

7. Graham, Douglas A., "Optimum Scheduling and Sequencing of Batch Processes," Master of Engineering report, University of Florida, August, 1970.

8. Hu, T. C., "Integer Programming and Network Flows," Addison-Wesley, Reading, Mass. (1969).

9. Gomery, R. E., "An Algorithm for Integer Solutions to Linear Programs," IBM Research Center, Research Report RC-189, January 29, 1960.

10. _____, "Outline of an Algorithm for Integer Solutions to Linear Programs," Bulletin of the American Mathematical Society, Vol. 64, pp. 275-278 (1958).

11. _____, J. Linear Algebra and Its Applications, 2, 4 (1969).

12. Ireland, Ronald J., "Solution Techniques for the 0-1 Linear Programming Problem," presented at the General Electric Applied Optimization Symposium, King of Prussia, Pennsylvania, Oct. 29-30, 1968.

13. Land, A. H., and Doig, A. G., Econometrica, 28, 3, 497-520 (1960).

14. Balas, Egon, Operations Research, 13, 4, 517-546 (1965).

15. Hadley, G., "Linear Programming," Addison-Wesley, Reading, Mass. (1962).

16. Greenberg, H. H., Operations Research, 16, 2, 353-361 (1967).

17. Trotter, L. E., "An Implicit Enumeration Algorithm for Integer Programming," Master of Science Thesis, Georgia Institute of Technology, September, 1970.

18. Shareshian, R., "Branch and Bound Mixed Integer Programming," IBM S/360 General Program Library 360D-15,2,005, 1968.

19. Driebeck, N. J., Management Sci., 12, 7 (1966).

20. Dzielinski, B. P. and Gomery, R. E., Management Sci., 11, 9 (1965).

21. Trauth, C. A. and Woolsey, R. E., Management Sci. 15, 9 (1959).

BIOGRAPHICAL SKETCH

Henry Xenophon Swanson, the son of Mr. and Mrs. Henry F. Swanson of Winter Park, Florida, was born on October 24, 1945.

He entered Atlanta's Georgia Institute of Technology in September, 1963. For the next four years, he worked alternate quarters as a co-op student for the Glidden Co. in Jacksonville, Florida. In June of 1968, Mr. Swanson received his B.S. degree in Chemical Engineering. He graduated on the Dean's List.

In September of 1968, Mr. Swanson entered the University of Florida under a National Defense Education Act Fellowship. He received his Master of Engineering degree in December, 1969, having chosen the Stability of Periodic Processes as his topic for research.

Mr. Swanson is presently employed as a process engineer for the Glidden-Durkee Co. in Jacksonville. He is married to the former Lynne Gregory of Atlanta.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Frank P. May

Frank P. May, Chairman
Professor of Chemical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Herbert E. Schweyer

Herbert E. Schweyer
Professor of Chemical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Robert D. Walker, Jr.
Professor of Chemical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

John B. Wallace
Assistant Professor of Management

This dissertation was submitted to the Dean of the College of Engineering and to the Graduate Council, and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

June, 1972

Dean, College of Engineering

A. G. Smith

Dean, Graduate School