Murphy, Frederic H.; Stohr, Edward A.

Working Paper

A Mathematical Programming Approach to the Scheduling of Sorting Operations

Discussion Paper, No. 164

Provided in Cooperation with:
Kellogg School of Management - Center for Mathematical Studies in Economics and Management Science, Northwestern University

Suggested Citation: Murphy, Frederic H.; Stohr, Edward A. (1977) : A Mathematical Programming Approach to the Scheduling of Sorting Operations, Discussion Paper, No. 164, Northwestern University, Kellogg School of Management, Center for Mathematical Studies in Economics and Management Science, Evanston, IL

This Version is available at:
https://hdl.handle.net/10419/220523

DISCUSSION PAPER #164

A MATHEMATICAL PROGRAMMING APPROACH TO THE SCHEDULING OF SORTING OPERATIONS*

by

Frederic H. Murphy

and

Edward A. Stohr

Revised March 1977

*The authors wish to thank Chuck Cooper of American National Bank for introducing us to the problem and Phillip Ryczek of Continental Bank for further assistance. The paper has also benefitted greatly from the comments of the Editor and referees.

ABSTRACT

A Mathematical Programming Approach to
the Scheduling of Sorting Operations

In this paper we describe an approach to the scheduling and/or real-time control of sorting operations in the presence of deadlines. The problem arises in the postal service where mail has to be sorted by zip codes and in the banking system where checks have to be sorted according to the bank on which they are drawn. In both applications losses are incurred if items miss their clearing deadlines. For example in check-sorting an extremely important objective of the control system is to reduce the 'float' i.e., the total dollar value of the checks which miss their deadlines. The proposed real- time control system utilizes a linear program which chooses between alternative sort-patterns and assigns the various processing steps to the time periods between deadlines.

1. Introduction

In this paper we are concerned with the design of optimal control systems for the sorting of documents by computer-controlled sorting machines. The problem has great economic importance since it occurs both in the postal service where mail has to be sorted by zip-code and in the banking system where checks have to be sorted by the bank in which they are deposited for return to the banks on which they are drawn. A discussion of the mail-sorting problem is given by Horn, [3], while a good description of an actual computer system for real-time control of check processing operations is given by Banham and McClelland, [2].

In both the postal and banking applications computer-controlled reader-sorters are employed which read the documents using either optical character recognition techniques or magnetic ink character recognition techniques. The documents are then directed by the machine to a particular pocket or hopper based on their identification code. Since the number of final destinations for the documents far exceeds the number of pockets available on the sorter many documents must be passed through the sorter more than once. Batches of documents arrive at random times through the day. The sorters group them according to their endpoint destinations. In the postal application the endpoints are associated with zip code regions; in the banking application the endpoints may be a collection of banks within a region, a Federal Reserve Bank, or a single bank which must be sent a large volume of checks. The sorting process is subject to a number of clearing deadlines and the performance of the system is closely related to the number and/or value of the documents which miss their deadlines on each day. For example, in check-sorting applications an important objective of the

processing system is to minimize the total dollar value of checks which miss their deadlines since one day's interest will be lost on these checks.

We now present some terminology and review the relevant literature. During processing, the items are read into a computer-controlled sorter. At each pass a code which translates to the endpoint is read off each item and the item is sent to a specified pocket. Sorters are available with different numbers of pockets. However, since the number of endpoints is substantially larger than the number of pockets, many items must go through multiple passes. This means that on early passes many endpoints have to be grouped into the same pocket, then broken down into subgroups and finally into individual endpoints. Since some endpoints have substantially higher volumes of items than others, it is clear that these endpoints ought to be separated before the low volume endpoints For a given batch of items (documents) containing n endpoints and a sorter with m pockets the sorting process can be represented by a tree. For example if m = 3 and the number of endpoints, n = 13, the tree may look like either of the trees in Figure 1 (the meaning of the letters shown above the leaves of the trees will be explained later).

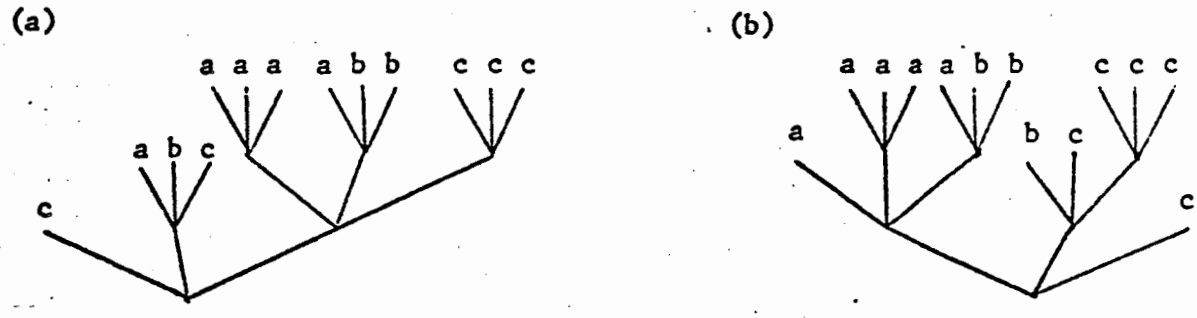

Figure 1

All nodes connected to a single arc are called 'exterior nodes' (Knuth [4]) and represent distinct endpoints for checks. All other nodes (the 'interior' nodes) represent 'rehandle' pockets i.e. a pass of a subgroup of checks through

the sorter. Since we can always add endpoints of zero volume, we need only consider m-ary trees, that is, trees where exactly m arcs emanate from each interior node. The number of interior nodes which must be added is given by the following Lemma.

Lemma 1: Let $i = n \bmod (m-1)$ and

$$j = \begin{cases} i & \text{if } i \geq 2 \\ m-1 & \text{if } i = 0 \\ m & \text{if } i = 1 \end{cases}$$

Then the number of zero volume endpoints to be added to make the m-ary tree is given by m-j.

For a proof of Lemma 1 see Knuth [4, p. 590]. From now on we assume that the m-j zero volume endpoints are added to ensure an m-ary tree.

Let v_i equal the expected volume of items for endpoint i and let w_i equal their expected value for $i = 1,2,\ldots,m$. We say that a tree is an h-level tree if the maximum number of arcs from the root to an endpoint is h. A node is at level h if it is the h^{th} node on the path from the root to this node. Note that an endpoint at level h goes through h-1 passes. For a given batch the processing procedure is completely specified by a sort-tree as given above together with the sequence of passes or visits to interior nodes.

An integer programming formulation of the problem of selecting a sort-tree (sort-pattern) to minimize the average number of passes per item is presented in Singh [8]. However the model becomes computationally intractable for practical problems since the number of integer variables increases very rapidly with the number of endpoints and the maximum number of levels specified in the tree. Nevertheless the formulation provides helpful insight. In a previous paper, [6], the authors have pro-

vided an algorithm which determines the class of sort-trees with minimum total processing time for a given batch. This algorithm works when the cost per pass of processing an item is positive. A restriction on the allowable number of levels in the tree can be included if required. The output of the algorithm is a tree of "Type 1" as shown in Figure 1(a) where the endpoints are ordered by increasing item volume from left to right, and the internal nodes are grouped to the right on each level.

Once a tree is chosen, a procedure to find the sequence of visits to the interior nodes which minimizes the average weighted time to visit each endpoint is given by Horn, [3]. However this objective function does not handle the case where there are specified deadlines for each endpoint. The sequencing of sorting operations on one machine is also discussed by Moore [5] who used simulation techniques to test standard scheduling procedures such as shortest operating time in a banking application. Here we assume that each endpoint has a single deadline. Let T_t be the time interval between the t-1st and the tth deadline. Define B_{it} to be the value of endpoint i if it is processed during the tth interval. For example, in a check-sorting application this value is a function of the interest rate, of the expected total value of checks for endpoint i within the batch and of whether the processing for endpoint i is completed before or after its deadline. In a postal application the value of the items for endpoint i may be defined as a weighted sum of the expected number of 1st class, 2nd class, and 3rd class mail items associated with the ith zip code area. Our objective throughout the paper is to maximize the total expected benefits.

In Section 2 we formulate the processing problem in terms of an integer program which extends the work of Singh [8] to allow for multiple batches and time sequencing of operations in the presence of deadlines. Although the formulation provides interesting insights into the combinatorial nature of the problem, it is computationally difficult. We therefore present a somewhat different linear programming approach to the problem in Sections 3 and 4. This approach can be easily implemented and can be used either to produce preplanned schedules or to provide real-time control. Finally, Section 5 provides an example illustrating the proposed procedures.

2. Integer Programming Formulation

The environment for our model is as follows. A number of batches of items arrive throughout the day. The arrival times and composition of the batches can be predicted with reasonable accuracy. The batches are to be processed using one or more sorters. For expositional purposes the sorters are assumed to have m pockets however the model can be generalized to allow for a multiplicity of sorter sizes. If there is more than one sorter the definition of T_t is modified so that it equals the total available sorter time between deadlines t-1 and t. The time to process a batch of items through one pass is composed of a fixed setup time, c, plus a variable time, λ, per item. For each batch, b, let B_{it}^{b} denote the benefit associated with isolating endpoint i in period t and let q_i^b denote the expected volume of items for endpoint i.

For expositional purposes we will assume that we wish to control the processing in a real-time mode, i.e. we monitor the system continuously and adjust our processing strategy accordingly. A pass of a batch through a sorter creates a set of completely separated endpoints and/or a new set of sub-batches to be sorted

subsequently. At any time during the day there is a set of batches waiting for processing to begin and another collection of partially sorted batches. No distinction need be made between these two types of batches since the sorting pattern for a sub-batch can also be represented by an m-ary tree. After each pass of a batch through a sorter a decision has to be made concerning the next batch to be processed and the sorting pattern to be used for the chosen batch. This problem is modeled by the following integer program. The solution of the integer program specifies the optimal sort pattern for each batch together with the time interval between deadlines in which each pass associated with an interior node is made. Processing may then commence on any batch assigned to the first time interval.

We define the following indices:

d_h = d^{th} interior node at level h

e_{h-1} = e^{th} interior at level h-1

f_{h-2} = f^{th} interior node at level h-2

and variables for each batch b:

$$x^b_{ie_{h-1}} = \begin{cases} 1 & \text{if the } i^{th} \text{ endpoint is assigned to the } e^{th} \text{ interior node at level } h-1 \\ 0 & \text{otherwise} \end{cases}$$

$$x^b_{ie_{h-1}t} = \begin{cases} 1 & \text{if } x^b_{ie_{h-1}} = 1 \text{ and the } i^{th} \text{ endpoint is scheduled to be isolated in the time period between the } t\text{-}1^{st} \text{ and } t^{th} \text{ deadline} \\ 0 & \text{otherwise} \end{cases}$$

$$v^b_{d_h e_{h-1}} = \begin{cases} 1 & \text{if the } d^{th} \text{ interior node at level } h \text{ is assigned to the } e^{th} \text{ interior node at level } h-1 \\ 0 & \text{otherwise} \end{cases}$$

$$v^b_{d_h e_{h-1} t} = \begin{cases} 1 & \text{if } v^b_{d_h e_{h-1}} = 1 \text{ and the pass associated with the } d^{th} \text{ interior node at level } h \text{ is scheduled for processing in the time period between the } t\text{-}1^{st} \text{ and } t^{th} \text{ deadline.} \\ 0 & \text{otherwise} \end{cases}$$

$z^b_{e_{h-1}}$ = maximum number of endpoints associated with the e^{th} interior node at level h-1

$\tau^b_{e_{h-1} f_{h-2}}$ = the total processing time for the batch associated with the interior node defined by $v^b_{e_{h-1} f_{h-2}}$

The objective is to maximize the total benefits received:

$$(1) \quad \max \sum_b \sum_t \sum_{e_{h-1}} \sum_i B^b_{it} \, x^b_{ie_{h-1}t}$$

subject to the following constraints.

A time constraint associated with the period between each deadline:

$$(2) \quad \sum_b \sum_h \sum_{d_h} \sum_{e_{h-1}} \tau^b_{d_h e_{h-1}} \, v^b_{d_h e_{h-1} t} \leq T_t$$

A requirement that each batch is processed in some time period:

$$(3) \quad \sum_t v^b_{d_h e_{h-1} t} = v^b_{d_h e_{h-1}}$$

A precedence requirement on sub-batches:

$$(4) \quad v^b_{d_h e_{h-1} t} \leq \sum_{k \leq t} \sum_{f_{h-2}} v^b_{e_{h-1} f_{h-2} k}$$

A requirement that each endpoint is processed in some time period:

$$(5) \quad \sum_t x^b_{ie_{h-1}t} \leq x^b_{ie_{h-1}}$$

Constraints which define the processing times for the non-endpoint nodes:

$$(6) \quad \tau^{b}_{e_{h-1}f_{h-2}} = \sum_{d_h} [\tau^{b}_{d_h e_{h-1}} - c] \, v^{b}_{d_h e_{h-1}} + \lambda \sum_{i} q^{b}_{i} \, x^{b}_{ie_{h-1}} + c$$

A requirement that each endpoint be isolated:

$$(7) \quad \sum_{h-1} \sum_{e_{h-1}} x^{b}_{ie_{h-1}} = 1$$

A requirement that the number of sub-batches and endpoints isolated at each pass equals the number of pockets.

$$(8) \quad \sum_{d_h} v^{b}_{d_h e_{h-1}} + z^{b}_{e_{h-1}} = m \sum_{f_{h-2}} v^{b}_{e_{h-1}f_{h-2}}$$

A requirement that the d^{th} interior node can be assigned to at most one non-endpoint node at the next lower level:

$$(9) \quad \sum_{f_{h-2}} v^{b}_{e_{h-1}f_{h-2}} \leq 1$$

A requirement that the number of endpoints isolated not exceed the number of pockets available after assignment of the sub-batches:

$$(10) \quad z^{b}_{e_{h-1}} \geq \sum_{i} x^{b}_{ie_{h-1}}$$

Finally, all variables are assumed to be non-negative.

Note first that constraints (2) and (6) are non-linear. To determine the sort pattern for even a single batch without considering the time of processing assuming 500 endpoints and a maximum of four passes per endpoint, we would need 2000 variables. With twenty time periods the number grows to 40,000 variables for describing a single batch. As there are numerous batches processed in each day this integer program is clearly impractical for either the planning or real-time control of the sorting system.

3. A Linear Programming Approach to Scheduling - Phase 1

To obtain a practical scheduling system we decompose the model described in the previous section into two phases. In the first, a set of candidate sort-patterns is generated for each batch according to the methods specified in [6] and described briefly below. Thus, in contrast to the formulation in Section 2, we no longer attempt to determine the optimal sort-patterns and the optimal processing sequence simultaneously. Although this simplifies the problem greatly, it would still be difficult to devise job-shop heuristics or rules which could adequately cope with the combinatorial nature of the problem and the kinds of trade-offs involved. In the second phase the candidate sort patterns are therefore incorporated in a linear program (described in Section 4) which chooses a sort-pattern for each batch from among those provided and assigns the processing of each interior node in the selected pattern to a time interval between deadlines. Ideally, the program should be solved every time a sorter becomes available for another pass. However, to reduce the computational burden a good compromise would be to run the program periodically or at the time of arrival of each new batch. As a further possibility, the program could be run periodically (say once per month) and used to set up target processing schedules--perhaps one for each day of the week. Thus a wide range of implementation possibilities exist from real-time control to pre-planned schedules. The latter possibility can be readily implemented since the algorithm for geeerating the alternative trees during phase 1 exists (see [6]), phase 2 involves only linear programming, and no changes in existing operational procedures would be required except for the substitution of the computed optimal sort-patterns and schedules. The implementation of some form of real-time control would be helpful because it allows the system to react to unexpected volumes of documents and other unexpected events. However, some adjustment to the existing computer and manual procedures would be required.

We assume that it is possible to maintain a data base of the expected volume, q_i, and value, v_i, of the items associated with each endpoint i in each batch on any given day of the week. These quantities can be updated using techniques such as exponential smoothing. Using this data base a collection of sort-patterns can be prepared periodically as described below. These patterns are in turn used to generate the data (concerning the processing times associated with each internal node, etc.) required by the program in phase 2.

A sort-pattern for a batch or sub-batch is represented by an m-ary tree and the linear program can be used with any arbitrary collection of sort-patterns. Thus, as is the case in most document sorting operations, only two or three patterns may be used - one for each major classification of batch types or division of the day (e.g. morning, afternoon and evening). This restriction of the number of sort-patterns reduces the complexity of tracing the sub-batches through the various processing stages and has been found useful in manual control systems. A greater diversity of sorting patterns is possible if the tracing operations are under computer control with instructions to the operators displayed on CRT devices. However, for computational reasons, it is still desirable to restrict the number of sort-patterns considered. One possibility is to use only sort-trees of "Type 2", which retain the property of minimum average processing time and which also have the property that the number of set-ups required to completely process all the checks for the most imminent deadline is minimized. Such a tree is shown in Figure 1(b) where the letters above each endpoint refer to the associated deadline with deadline 'a' being the most urgent and deadline 'b' the next most urgent and so on. A Type 2 tree is obtained from a Type 1 tree as follows. Order the endpoints at each level in the Type 1 tree from left to right by increas-

ing time to deadline and within the group of endpoints for each deadline by decreasing value. Next associate each internal node with the highest priority deadline in the sub-tree for which it is a root and move it to the left until an endpoint with the same or higher priority deadline is encountered. Note that the endpoints associated with the different deadlines tend to be grouped together. Similarly, endpoints with high expected dollar value tend to be grouped together. Also note that the number of possible Type 2 trees for each batch equals the number of deadlines associated with its endpoints.

To summarize, the procedure in Phase 1 is to use data concerning the expected volume of items for each endpoint to generate a minimum total processing time (Type 1) tree for each anticipated batch or set of batches of incoming items. This is done using the algorithm given in [6]. The Type 1 tree for each batch is used to generate a number of Type 2 trees--one for each interval between time deadlines. This forms a basic set of alternative sort-patterns for each batch. To this basic set management can add other likely candidates as required.

4. Linear Programming Approach to Scheduling - Phase 2

In this section we assume that the first phase of the procedure has been executed and that a suitable set of sort-patterns (m-ary trees) is available. We now present the integer linear programming model which is to be used in the second phase.

We define the following indices:

r = r^{th} pattern for a given batch

s = s^{th} interior node in a sort-pattern for a batch where s = 1 denotes the root of the m-ary tree and the other interior nodes are enumerated from left to right on successive levels in the tree.

The data for the problem includes the time intervals between deadlines, T_t, as before. In addition, the sort-patterns generated during the first phase provide the following information:

B^b_{rst} = the benefit obtained from isolating the endpoints associated with the s^{th} interior node of the r^{th} pattern for batch b in the time interval between the $t\text{-}1^{st}$ and t^{th} deadline.

σ^b_{rs} = the total processing time associated with the s^{th} interior node of the r^{th} pattern for batch b. (This can be calculated using an equation similar to (6)).

$I^b_{(r,s)}$ = the set of interior nodes that are directly connected to the s^{th} interior node in the r^{th} pattern for batch b.

We define the following variables:

$$v^b_{rst} = \begin{cases} 1 & \text{if interior node, s, of the } r^{th} \text{ pattern associated} \\ & \text{with batch b is processed in time period t} \\ 0 & \text{otherwise} \end{cases}$$

As before, the objective is to maximize total benefits:

$$(11) \quad \max \sum_b \sum_r \sum_s \sum_t B^b_{rst} v^b_{rst}$$

subject to the following constraints.

Time constraints:

$$(12) \quad \sum_b \sum_r \sum_s \sigma^b_{rs} v^b_{rst} \leq T_t \qquad \text{for all } t$$

Constraints arising from the precedence relations in the sort-patterns:

$$(13) \quad \sum_{u \leq t} v^b_{rsu} \geq \sum_{u \leq t} v^b_{r\hat{s}u} \;, \; \hat{s} \in I^b(r,s) \qquad \text{for all } t, b$$

A requirement that all interior nodes be sorted and that a single pattern is chosen for each batch:

$$(14) \quad \sum_r \sum_t v^b_{rst} = 1 \qquad \text{for all } s,b$$

The constraints (14) operate in the following manner. For $s = 1$ the constraints (13) and (14) select a single pattern for each batch from among those provided. For each pattern selected we wish to ensure that every interior node is sorted. Let $\hat{r}$ be a selected pattern for batch b. From (13) $v^b_{rst} = 0$ for $r \neq \hat{r}$. This means that equations (14) are equivalent to:

$$(15) \quad \sum_t v^b_{\hat{r}st} = 1 \qquad \text{for all } s$$

By the following Lemma these constraints guarantee that each endpoint is isolated.

Lemma 2: For a given number of endpoints, all sort-patterns contain the same number of interior nodes.

Proof: We have n external nodes and (say) N internal nodes. The number of arcs in the tree is then n+N-1, but since every internal node has m outwardly directed arcs, we see that $mN = n+N-1$ and $N = \frac{n-1}{m-1}$ which is constant.

Lemma 2 implies that there is the same number of constraints of the form (15) for each sorting pattern. Therefore the constraints (14) guarantee the completion of the sort-pattern for each batch.

It would be economically infeasible to solve this integer program on a repetitive basis. We therefore solve it as a linear program and provide decision rules as discussed below for dealing with any non-integer variables. To further reduce computation and storage requirements we note that the constraints (14) are generalized upper bound constraints and that special computational techniques can be applied (see Agbadudu, [1]). This is especially relevant if the program is to be used for real-time control. In this context note that at the time the linear

program is run a number of batches and partially processed sub-batches will exist and it will be necessary to include the relevant data for these batches. As in the paper by Reiter [7] it is also possible to anticipate future batch arrivals by including artificial batches,b, with B^b_{rst} equal to zero if the batch is not expected to arrive during the current period, t.

Computational experience shows that there are relatively few non-integer variables in the final solution of the linear program. The non-integer variables which do appear come from two sources. The first is where more than one Type 2 sort-pattern is chosen for a batch. In this case it is necessary to select one of the alternative patterns. A single pattern can be assigned to each batch, b, by choosing the pattern, r, with the maximum value of $\Sigma_t v^b_{rlt}$. After this process has been carried out for all batches, b, the linear program can be rerun starting from the previous optimal solution and with v_{rlt} set equal to zero for the eliminated patterns.

In the new optimal solution to the linear program the only possible source of non-integer variables is the splitting of the processing of interior nodes into several time periods. Since it is unlikely that the processing times for the interior nodes will sum exactly to T_t, one node may, in any case, be split between time periods without loss. A process for generating a reasonable schedule for the processing of the interior nodes in the current period (t = 1) which also resolves any possible ambiguity arising from non-integrality is as follows:

(1) Initially consider only nodes with $v^b_{rs1} = 1$ in the optimal solution.

(2) These nodes define a set of subtrees. For each internal node in these sub-trees compute the difference in benefits between processing the node in the current period and in the succeeding period.

(3) Assign to each node the sum of the values obtained in step (1) for this node and all of its successor nodes (i.e. nodes in the sub-tree which cannot be processed before this node).

(4) For each node divide the quantity calculated in step (2) by the total time required to process the node.

(5) Successively process the root nodes of the sub-trees (and resulting sub-trees, etc.) always selecting the root node with the highest value of the ratio computed in step 3 to be processed next.

Using this procedure the nodes with the greatest benefit per unit time will be scheduled first. This is beneficial if there is a positive probability of a delay or breakdown in the processing system. After this process has been carried out a certain amount of time will be left before the first deadline for processing the internal nodes with $0 < v^b_{rs1} < 1$ in the optimal solution. To compute a tentative schedule for these nodes steps (2) to (5) are repeated except that the values computed in step (2) are multiplied by the associated values of v^b_{rs1}. An example of this process is given in the next section. If the linear program is solved frequently, the losses due to this approximation procedure for dealing with non-integer solutions will be reduced. For example, if the procedure is used for real-time control and a new optimal solution is computed every time a sorter becomes available for another pass it is only necessary to select the next (sub) batch to be processed.

5. Illustrative Example

For purposes of illustration we assume a processing horizon with three deadlines indexed 1, 2 and 3. Processing is allowed to occur after the third deadline by replacing the equality constraints in (14) by less than or equal to constraints. Batches 1 and 2 are assumed to be ready for processing at the start of the first time interval. Batches 3 and 4 are anticipated to arrive at the beginning of the second time interval and batches 5 and 6 are

anticipated to arrive at the beginning of the third time interval. For simplicity the data for the batches is divided into two categories with batches 1, 3 and 5 belonging to the first category and batches 2, 4 and 6 belonging to the second. The endpoint volumes and associated deadlines are shown in Table 1.

TABLE 1

Endpoint Volumes and Deadlines

	Endpoint												
	1	2	3	4	5	6	7	8	9	10	11	12	13
Category 1													
q_i	200	100	100	50	20	20	20	10	10	10	1	1	1
Deadline	3	1	2	3	1	1	1	1	2	2	3	3	3
Category 2													
q_i	200	200	200	100	100	100	80	20	10	10	5	2	1
Deadline	2	2	1	3	2	1	1	2	2	1	3	2	1

The alternative Type 2 sort-patterns for the batches are shown in Figure 2. Here the Type 2 sort-trees are identified by their category number and the time deadline given the highest priority during their construction by the method outlined in Section 1. Thus, sort-tree 1-2 is associated with category 1 endpoint data and time interval 2 (in this tree the number of set-ups required to separate all the endpoints with the second deadline is minimized). In Figure 2 the internal nodes are numbered from left to right on successive levels. Two numbers are written above each endpoint node. The lower number is the endpoint index and the upper number is the associated deadline. In the linear program the relevant sort-patterns for the first two time periods were specified as alternatives for batches 1 and 2 which arrive in the

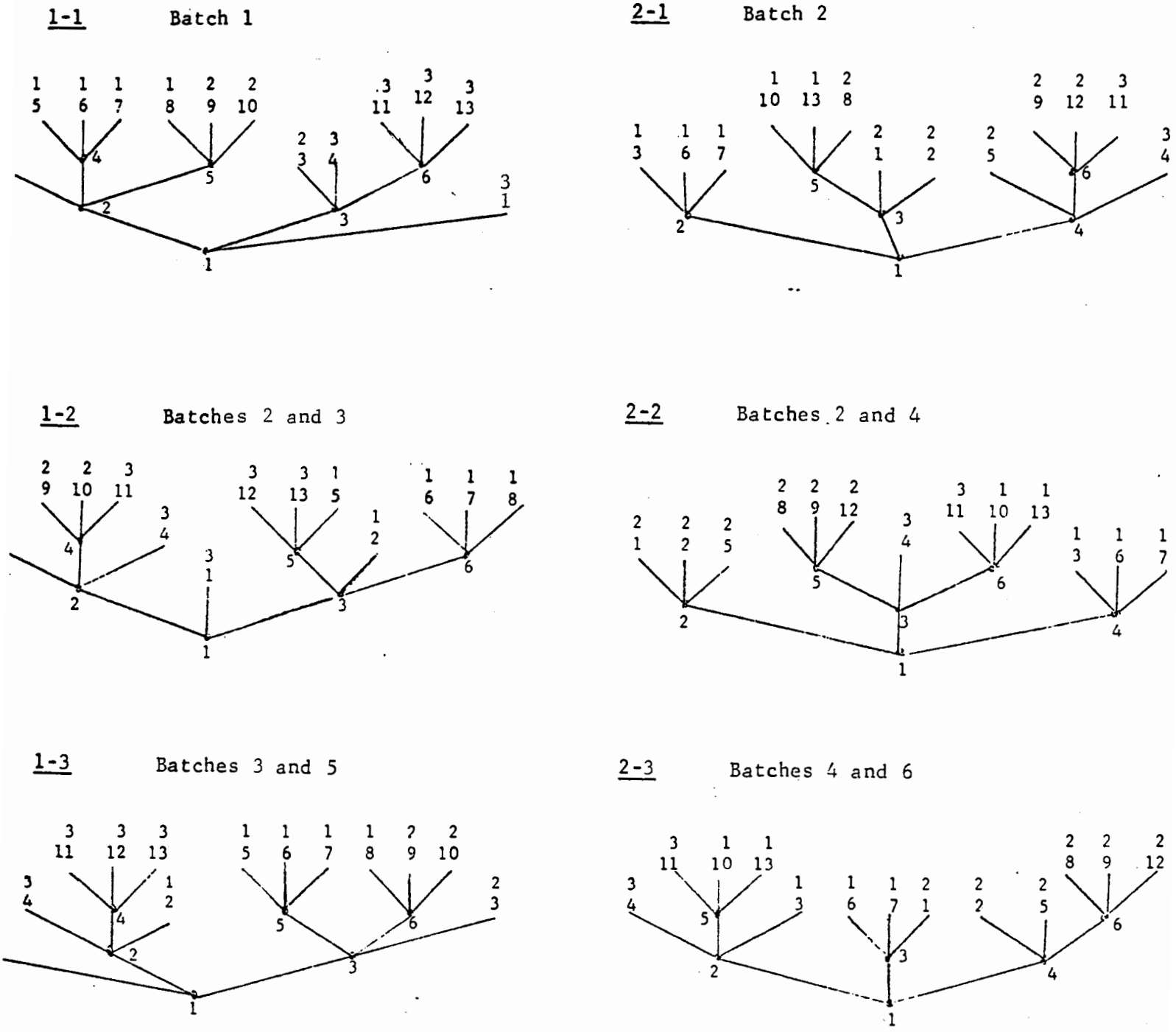
1-1 Batch 1
2-1 Batch 2
1-2 Batches 2 and 3
2-2 Batches 2 and 4
1-3 Batches 3 and 5
2-3 Batches 4 and 6

Figure 2

first time interval. For these batches the sort-pattern index, r, has values 1 and 2 corresponding respectively to the sort patterns for the first two time periods. Similarly, the relevant sort-patterns for time periods two (r=1) and three (r=2) were specified as alternatives for batches 3 and 4 while only the sort-patterns for period three (r=1)were specified as alternatives for batches 5 and 6. To illustrate the indexing scheme, variable v^4_{123} is associated with the second internal node of the first pattern for batch 4 (sort-tree 2-2). If $v^4_{123} = 1$ in the optimal solution to the linear program, then batch 4 is processed using the sort-pattern 2-2 and node 2 is isolated in period 3.

In the illustrative example, the benefit for separating endpoint i of batch 6 in period t equals q^b_i if t is less than or equal to the deadline for endpoint i while it equals zero otherwise. Also, the average time to process an item through one pass, $\lambda = 1$, and the set-up time for a pass, $c = 10$. Using this information the values of B^b_{rst} and σ^b_{rs} are readily computed. For example, $B^4_{152} = 32$ and $\sigma^4_{15} = 42$ (see Table 1 and sort-tree 2-2 in Figure 2). Finally, the time intervals for deadlines 2 and 3 are assumed to be $T_2 = 3000$ and $T_3 = 3000$. The linear program for this problem has 149 constraints and 132 variables. The optimal solutions for two different time intervals, before the first deadline are shown in Table 2.

The time constraints for periods 1 and 2 are tight while that for time period 3 is slack in the optimal solutions for both $T_1 = 2500$ and $T_2 = 2000$. As might be expected the reduction in time available before the first dead-

TABLE 2

Solutions to the Linear Program

Batch	$T_1 = 2500$		$T_1 = 2000$	
	Non-zero Basic Variable	Value	Non-zero Basic Variable	Value
1	$v^1_{111}, v^1_{121}, v^1_{141}, v^1_{151}, v^1_{163}$	1	v^1_{222}, v^1_{242}	.1
	v^1_{131}	.15	$v^1_{211}, v^1_{231}, v^1_{251}, v^1_{261}$	.69
	v^1_{132}	.85	$v^1_{212}, v^1_{233}, v^1_{253}$	.31
2	$v^2_{112}, v^2_{231}, v^2_{241}, v^2_{261}, v^2_{222}, v^2_{252}$	1	$v^2_{211}, v^2_{241}, v^2_{222}, v^2_{232}, v^2_{252}, v^2_{263}$	1
3	$v^3_{112}, v^3_{122}, v^3_{142}, v^3_{133}, v^3_{153}$	.73	$v^3_{213}, v^3_{223}, v^3_{243}$	.79
	$v^3_{213}, v^3_{223}, v^3_{243}$	.27	$v^3_{112}, v^3_{122}, v^3_{142}, v^3_{133}, v^3_{153}$	.21
4	$v^4_{112}, v^4_{122}, v^4_{132}, v^4_{152}, v^4_{163}$	1	$v^4_{112}, v^4_{122}, v^4_{132}, v^4_{152}, v^4_{163}$	1
5	$v^5_{113}, v^5_{123}, v^5_{143}$	1	$v^5_{113}, v^5_{123}, v^5_{143}$	1
6	$v^6_{113}, v^6_{123}, v^6_{153}$	1	$v^6_{113}, v^6_{123}, v^6_{153}$	1

line increases the scatter of the processing of the nodes from a given sort-tree amongst the various time intervals. It can be seen from Table 2 that the Type 2 sort-patterns associated with the second deadline were optimal for the period one batches in three out of four cases. In two of these cases, however, the period one sort-patterns represented alternative optima. The grouping of endpoints by deadline in the Type 2 trees is usually advantageous. For example, the linear program used in the illustration was run 16 times with Type 1 sort-trees as alternative patterns for batches 1 and 2 (for values of T_1 = 1000, 1500, 2000, 2500 and with C = 10 and C = 100 for each value of T_1). Type 1 sort-trees were selected in preference to Type 2 sort-trees in only two of these cases.

We now use the results of the linear program to compute a tentative processing schedule for time period 1 by the method described in Section 4. Since a unique pattern was chosen for the batches arriving during the first time interval, there is no need to adjust the solution in this respect. From Table 2 it can be seen that internal nodes 1 and 4 of sort-tree 2-2 in Figure 2 are the only nodes to be processed completely in period 1 according to the optimal solution of the linear program. From the precedence relationship node 1 must be processed first followed by node 4. The time taken for the two set-ups and to pass all the items in batch 2 through the first pass (node 1) and the items for endpoints 3, 6 and 7 in the second pass (node 4) is 1428 time units. Since v^1_{2s1} = .69 for s = 1, 3, 5, 6 nodes 1, 3, 5 and 6 of sort-tree 1-2 in Figure 2 become candidates for scheduling during the remainder of period 1. The calculations to determine the best sequence of visits to these nodes according to the method given in the last section are set out in Table 3.

TABLE 3

Scheduling Batch 1 in Time-Interval 1

	(1)	(2)	
Node s	σ^1_{2s}	Difference in Benefits	$v^1_{2s1} \times$ (2) ÷ (1)
1	553	170	.21
3	182	170	.64
5	32	20	.43
6	60	50	.58

From the last column in Table 3 the tentative processing sequence for batch 1 is to visit internal nodes 1, 3, 6 and 5 of sort-pattern 2 in that order. However, only node 1 can be fully processed before the end of the first period and for a real-time control application it would obviously be desirable to rerun the linear program with updated information before that time. Thus it may only be necessary to invoke the heuristic procedure for resolving ambiguities caused by non-integral solutions when the time before the first deadline is short.

6. Conclusion

In this paper we have described a linear programming approach to the scheduling and/or real-time control of sorting operations in the presence of deadlines. The linear program chooses between alternative sort-patterns and assigns the passes involved to the time periods between deadlines. Although any sort-patterns may be used as data we have suggested that candidate sort-patterns be chosen from the class with minimal total processing time and that allowance be made for the presence of deadlines by regrouping the internal nodes by the method described in Section 3.

Although we have described a straightforward and intuitively reasonable approach to the scheduling problem it is apparent that there is room for experimentation with the choice of a suitable data base of sorting patterns, the determination of the times at which the linear program is run, and the rules for making use of the linear programming solution. These decisions might be tested and improved by incorporating the suggested algorithms for generating alternative sort-patterns and the linear programming approach to the selection of the next batch to be processed, as components in a larger simulation program.

References

1. Agbadudu, Amos, "Generalized Upper Bound, Variable Upper Bound and Extensions for Large Scale Systems," unpublished Ph.D. dissertation, Graduate School of Management, Northwestern University.

2. Banham, J.A. and McClelland, P., "Design Features of a Real-Time Check Clearing System," IBM Systems Journal, No. 4, 1972.

3. Horn, W.A., "Single-Machine Job Sequencing With Tree-like Precedence Ordering and Linear Delay Penalties," SIAM Journal of Applied Mathematics, Vol. 23, No. 2, September, 1972.

4. Knuth, D.E., The Art of Computer Programming, Vol. 1, Addison-Wesley, Reading, Massachusetts, 1969.

5. Moore, L.J., An Experimental Investigation of a Computerized Check Processing System in a Large City Bank Using Digital Simulation, Ph.D. Thesis, Arizona State University, September, 1970.

6. Murphy, F.H. and Stohr, E.A., "A Dynamic Programming Algorithm for Check Processing," Management Science (to appear).

7. Reiter, S., "A System for Managing Job-Shop Production," University of Chicago, Journal of Business, July, 1966.

8. Singh, B.J., "A Heuristic Approach to Solve a Large Scale Linear Programming Problem," presented at the ORSA-TIMS Conference, Fall, 1974.