DISCUSSION PAPER NO. 158

A DYNAMIC PROGRAMMING ALGORITHM FOR CHECK SORTING*

by

Frederic H. Murphy

and

Edward A. Stohr

Revision: December, 1976

*The authors wish to thank Chuck Cooper of American National Bank for presenting the problem to us and Phillip Ryczek and Bob Malczewski of Continental Bank and Richard Rauscher and Bob Maychec of First National Bank for further assistance. The paper has benefited greatly from the constructive comments of the editor and referees.

ABSTRACT

The motivation for this paper is a problem faced by banks which process large volumes of deposited checks. The checks must be separated by bank number before shipment to the Federal Reserve or other banks. The sorting is usually accomplished using a reader-sorter which reads the magnetic ink characters on the checks and separates them into different "pockets". This paper characterizes the optimal sorting strategy and describes an efficient procedure for finding the optimal solution for problems of the size generally found in practice. The algorithm is based on a two state dynamic programming recursion in wh ich characterization theorems are used to drastically reduce the size of the state space and in which the storage requirements are minimal. The paper includes an analysis of computational experience and describes how the algorithm can be used in a real time environment with deadlines.

1. Introduction

Check processing is a major function of the commercial banking system and the total annual cost of these operations exceeds three billion dollars (L. Moore [5]). An important aspect of the check-processing problem is the optimization of the system which sorts checks by their endpoint destinations. 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. For checks drawn on the bank itself the endpoints are the banks own checking account customers. In one large Chicago bank approximately two million checks with a total value of roughly $1 billion dollars are sorted to 400 different end-points on a typical day. A description of the problems faced by the Bank of England in designing a computer system for the real time control of their check processing operation is given by Banham and McLelland [1].

Large check sorting operations involve complex man-machine systems whose reliability and efficiency are crucial to the successful operation of the bank and to the maintenance of good relations with its customers. The objectives of the system differ somewhat depending on the circumstances and the class of check involved. The checks drawn on the bank itself must be separated from the other checks and sorted by account number. For these checks a suitable objective is simply to minimize the total processing time a ('throughput' criterion). On the other hand, checks drawn on other banks must be returned to those banks either directly or via a clearing bank in the Federal Reserve system. In either case checks for the various endpoints will be subject to clearing deadlines. Essentially the bank loses one day's interest on all checks which miss their deadlines (at the current Federal Funds rate a loss of approximately $150 would be incurred for each $1 million of checks not processed on time). For this class of checks the processing system must therefore be designed to allow for the presence of deadlines and for the expected

(or actual) value of the checks associated with the various endpoints as well as for efficient throughput. For given processing hardware the most important decisions affecting performance involve the choice of one or more 'sort-patterns' or strategies for separating the checks according to the various endpoints.

During processing, the checks are read into a computer-controlled sorter. At each pass a code which translates to the endpoint is read off each check and the check 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 checks must go through multiple passes. This means that on early passes many endpoints have to be grouped into the same pocket, broken down into subgroups and finally into individual endpoints. Since some endpoints have substantially higher volumes of checks than others, it is clear that these endpoints ought to be separated before the low volume endpoints. To represent the separation process, the sorting pattern can be described as a tree with directed arcs. For example, a tree for a four pocket sorter with 19 endpoints could be as shown in Figure 1(a).

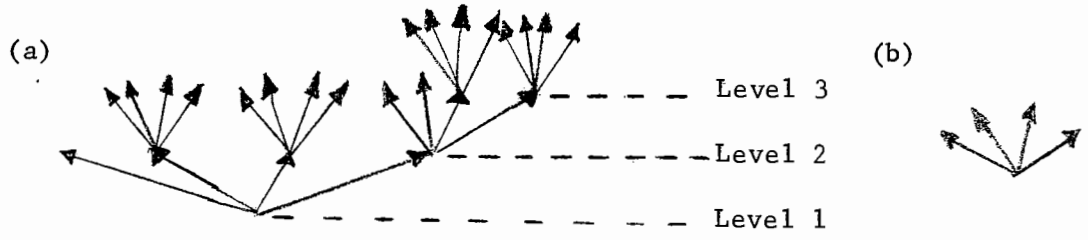


FIGURE 1

Nodes connected to a single arc are called 'exterior nodes' and all other nodes are called 'internal nodes' (Knuth [4, p. 399]). The external nodes represent distinct endpoints or 'kill-pockets' for checks. The root node represents the first pass of a batch of checks through the sorter while the other internal nodes represent 'rehandle' pockets in the check sorting process. 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 $h \leq H$ if it is the h^{th} node on the

path from the root to this node. Note that checks assigned to an endpoint at level h go through h-1 passes.

The problem of optimizing the check sorting operation can be thought of basically as that of choosing one or more sort-trees for use with the different classes of incoming checks. When the processing is subject to deadlines it is also necessary to simultaneously consider policies for sequencing the processing of the various batches and sub-batches associated with the internal nodes of the trees. As far as is known, the sort trees used in practice are developed heuristically. One Chicago bank uses only one sort-tree while another uses six with the choice of sort-tree depending on the time of day. In the latter case the sort-trees all have the same general shape and differ only because the endpoints are assigned to different pockets. In this paper we are primarily concerned with the development of a technique for generating sort-trees which are optimal with respect to a throughput criterion. However, in the last part of the paper, we show how the sort-trees generated by our algorithm can be modified without increasing the total processing time to allow for the presence of processing deadlines. The modified trees can be used in situations where it is necessary to minimize losses from missed deadlines. In a subsequent paper [6], we show how these trees can be used as inputs to a dynamic scheduling algorithm.

2. Problem Formulation

Let the sorter have m pockets to process checks for n endpoints.[1] The volume of checks to endpoint i, for i = 1,...,n has mean q_i. Without loss of

---

[1] The value of m does not include the reject pocket(s) necessary for unreadable items.

generality we order the endpoints so that $q_i \geq q_{i+1}$ for $i = 1,\ldots,n$. For a given sort tree define p_i as the number of passes for the checks of endpoint i. Let c_h be the cost of sorting a check through h passes. We assume $c_h < c_{h+1}$. Nonlinearity of the costs allows for the probability of check damage increasing more than proportionately with the number of passes beyond the first. One interpretation of the costs c_h is as follows. Let d be the average variable time to pass a check through the sorter and let r_h be the probability of a check being rejected (unreadable) on the h^{th} pass. Assume that checks which are rejected are subject to special handling procedures which on the average involve δ time units of processing per check. If the objective is to minimize the total processing time for both rejects and non-rejects, the expected time to sort checks which are separated on the h^{th} pass is:[1]

$$c_h = \sum_{k=1}^{h} \prod_{i=1}^{k} (1-r_i)d + \sum_{k=1}^{h} r_k \prod_{i=1}^{k-1} (1-r_i) \cdot \delta$$

which is an increasing function of h. In the first part of the paper our objective is to minimize the total cost, $\sum_i c_{p_i} q_i$. As shown later, a set-up cost associated with each internal node, which results from the physical handling of the checks, does not affect the optimal solution because the number of internal nodes is constant. In practice it is often necessary to isolate certain classes of documents on the first pass through the sorter. Possibilities here include travellers' checks which damage and checks with very high dollar values. This requirement is accommodated in the computer algorithm to be described simply by allowing the sorter to have less than m pockets available on the first pass and eliminating the special endpoints from further consideration.

An integer programming formulation of the check-sorting problem for the case where $c_{p_i} = cp_i$ is presented in Singh, [7]. However, a thousand endpoint problem leads to six thousand integer variables and 1006 constraints

---

[1] Note that d and δ could alternatively have been defined as dollar cost per item pass and per item respectively.

if a maximum of 6 passes is expected for any endpoint. The approach we take is to construct and characterize a restricted class of trees that contains an optimal tree and to use a recursion to implicitly enumerate all the members of this restricted class. This recursion is based on a two-state dynamic program. By further characterization of the structure of the dynamic program we eliminate the need for a complete enumeration of all of the elements of the two dimensional state space, drastically cutting down the size of the problem.

Although our discussion is in terms of the check processing problem which motivated the study, the algorithm developed has several potential areas of application. The most obvious applications occur in other mechanical sorting processes for example, in the sorting of mail by zip code (Horn, [2].) In coding theory, the case where $c_{p_i} = cp_i$ has been solved by Huffman, [3]. In this context the algorithm to be described below will allow the generation of optimal codes where there is an increasing cost for the number of symbols used in a string. Such a cost structure might occur when humans are involved in the transmission or reception of strings of characters because longer strings are more difficult to comprehend leading to a longer processing time per symbol. Also, in contrast to Huffman's algorithm, the algorithm described here works when there is a restriction on the number of levels in the trees.

We now state some straightforward results which simplify the analysis by reducing the variety of trees which need be considered. Reassigning end-points (banks) to different pockets on the same level in the tree does not change the cost. Therefore, we can index the external nodes of the tree from left to right be decreasing volume and need only consider trees where the length of the path from the root to external node i is less than or equal to the length of the path from the root to external node j for $i < j$.

Furthermore, 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 nodes to be added is determined as follows (see Knuth [4, p. 590]):

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. In addition, the m-ary trees considered 'feasible' will have a form similar to that in Figure 1a with the longer paths from the root node to the endpoints skewed as far to the right as possible. From now on we assume that the m-j zero volume endpoints are added to ensure an m-ary tree.

For the sorting problem 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 $N = \frac{n-1}{m-1}$. Thus the number of internal nodes in any m-ary tree with n external nodes is constant and we need not consider the set up cost associated with each internal node

3. The Implicit Enumeration Algorithm

A formulation of the sorting problem can be stated in dynamic programming form in terms of a backwards recursion starting at maximum allowable height, H, of the tree and iterating down to level 1. The dynamic program is used to prove certain properties of the trees; these properties form the basis for the algorithm to be described later. The state-space of the problem at level h consists of all pairs of the form (i,r) where i indicates that the set of endpoints $\{i,i+1,\ldots,n\}$ have not yet been separated at level h of the sort

tree and r is the number of internal nodes at the level h under consideration. For example, at level h=3 in Figure 1a, i = 12 and r = 2.

The recursion presented is not the most 'natural' in the sense that some of the costs associated with 'future' actions are charged against 'current' costs. This is done to simplify the proofs of the characterization theorems which appear later.

Let:

(1) $f_h(i,r)$ = the minimum cost of the forest (that is, collection of trees - see [4, p. 306]), containing endpoints i through n with r roots at level h and with $p_j \geq h$ for $i \leq j \leq n$,

$$= \sum_{k=i}^{n} (c_{p^*_k} - c_{h-1})q_k, \quad 1 \leq h \leq H$$

where p_k^* is the number of passes for endpoint k in the minimum cost solution. That is, f is the minimum cost of sorting endpoints i through n from the h^{th} pass on, given that they have not been sorted out by their $h\text{-}1^{st}$ pass through the sorter. The recursive equation is:

$$(2) \qquad f_h(i,r) = \min_{j_1(i) \leq j \leq j_2(i)} f_{h+1}(j, mr-j+i) + (c_h - c_{h-1}) \sum_{k=i}^{n} q_k$$

To define j_1 and j_2 let $\rho = \min(rm, \frac{n-rm+i-1}{m-1})$ be the maximum feasible number of internal nodes at level h+1 for given i; then $j_1(i) = i+rm-\rho$ and $j_2(i) = i+rm-1$. Let i' be the smallest index endpoint at level h, and r' the number of internal nodes at level H-1 in the initial H-1 level tree. Then the terminal optimal value function is: $f_H(i',r') = \sum_{k=i'}^{n} (c_H - c_{H-1})q_k$. Note that we do not include the cost for the first h passes for endpoints i through n in $f_{h+1}(i,r)$. By including these costs as they are incurred in the current

cost component ($(c_h - c_{h-1}) \sum_{k=i}^{n} q_k$ at level h) we remove the effect of our decision variable, j, on the current cost. As a result we need analyze only one component of the recursion to draw conclusions about the structure of the dynamic program. This recursion is numerically intractible for any practical problem because of the two-dimensional state-space and the ranges for the variables. We therefore propose another enumeration scheme. The dynamic programming formulation (1) and (2) is utilized in the Appendix which gives the theoretical justification for the algorithm described below.

To simplify the discussion we define a plume to be a set of m arcs and m+1 nodes of which m nodes are exterior nodes. The arcs connect the external nodes to the remaining node which we call the root of the plume. A plume is shown in Figure 1(b) for the case m=4. We define the level of a plume to be the level of its root.

Let a_i be the number of internal nodes at level i and for an h-level sort tree define a 'sort vector' by $a = (a_1,\ldots,a_h)$. Note that $a_1 = 1$. The cost to complete the sort using the strategy defined by sort-vector, a, is denoted by C(a). To enumerate all trees of a given level, h, we could start with the lexicographically largest vector, a, which is feasible (i.e. has $a_1 = 1$ and $a_{i+1} \leq ma_i$, i = 1,2,...,h-1) and enumerate all feasible trees in a lexicographically decreasing order. Geometrically, the movement to the next tree in the sequence involves moving a plume (or plumes) from one level in the tree to the next highest level that is feasible (maintains an m-ary tree of the form shown in Figure 1a). Note that the plume(s) will now be associated with different endpoints since the exterior nodes are always maintained in decreasing order by check volume. Also, moving a plume up a level adds another exterior node at the lower level.

We now describe an implicit enumeration scheme for computing an optimal sort-tree with level less than or equal to H. H' will represent the current height of the tree. We start with the minimum height m-ary tree (lexicographically largest sort vector). We repeatedly increase H', the level of the tree, by one. At each value of H' we find a minimum cost tree with a maximum number of levels less than or equal to H'. We stop at $H'=H$ (or earlier if an increase in H' does not change the optimal tree). For a given H' the minimization proceeds downward from the top with the index k indicating the lowest level in the tree to which the optimization has extended. Each time we reduce k by 1 we have to reoptimize at all higher levels in the tree. The algorithm generates a lexicographically decreasing sequence of sort-vectors (but, as shown in Section 4, only a small fraction of the total possible number).

Algorithm[1]

1. Start with the lexicographically largest sort-vector. Let H' be the height of this tree and let it be represented by $a_1,a_2,\ldots,a_{H'}$. Set $k = H'-1$ and $b'_{H'} = a_{H'}$.
2. The optimal sort-vector for $a_1,\ldots,a_k$ fixed is $a_1,\ldots,a_k,b'_{k+1},\ldots,b'_{H'}$. If feasible, find $a_1,\ldots,a_k-1$, $b_{k+1},\ldots,b_{H'}$, the optimal sort-tree given $a_1,\ldots,a_k-1$ and go to step 3. If infeasible set k=k-1. If k=1 go to step 4; otherwise set $b'_{k+1}= a_{k+1}$ and repeat this step.
3. If $C(a_1,\ldots,a_k - 1,b_{k+1},\ldots,b_{H'}) < C(a_1,..,a_k,b'_{k+1},..,b'_{H'})$ set $a_k = a_k-1$ and $b'_{k+1} = b_{k+1},\ldots,b'_{H'} = b_{H'}$ and go to step 2. Otherwise set $b'_k= a_k$ and $k = k-1$. If $k > 1$ go to step 2; if $k = 1$ go to step 4.

[1] A number of details are omitted. A full description of the algorithm is available from the authors on request.

4. If $H' = H$, or $b'_{H'} = 0$ stop, $a_1, b'_2, b'_3, \ldots b'_{H'}$ is the optimal sort-vector. Otherwise let $a_1, a_2, \ldots, a_{H'+1}$ be obtained from $a_1, \ldots, a_k, b'_{k+1}, \ldots, b'_{H'}$ by removing a plume from the highest possible level and placing it at $H'+1$. Set $H' = H'+1$, $k = H'-1$ and let $b'_{H'} = 1$; go to step 2.

Detail of Step 2:

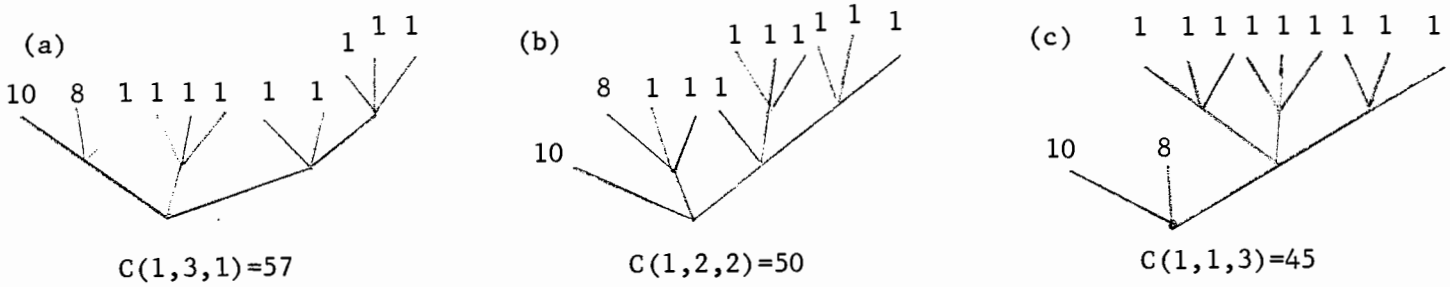
To obtain $b_{k+1}, \ldots, b_{H'}$ from $b'_{k+1}, \ldots, b'_{H'}$ in step 2 we recursively execute a procedure similar to step 2 for levels $h = H', H'-1, H'-2, \ldots, k+1$. The lexicographically largest sort-vector which must be considered at each level of optimization is obtained starting from $a_1, \ldots, a_k, b'_{k+1}, \ldots, b'_{H'}$ the optimal sort-vector given $a_1, \ldots, a_k$. As proved in the Appendix only one plume need be added at each level. Thus, if feasibility considerations permit we start the reoptimization at level k+1 from $a_1, \ldots, a_k-1, b'_{k+1}+1, b'_{k+2}, \ldots, b'_{H'}$ and at level k+2 from $a_1, \ldots, a_{k-1}, b'_{k+1} b'_{k+2}+1, \ldots, b'_{H'}$ etc. Thus this step is initialized by successively moving a plume to higher levels until H' is reached.

The algorithm is based on several 'convexity' results, the formal proofs of which are given in the Appendix. Theorem 1 shows that when we enumerate feasible sort trees in lexicographically decreasing order and when the only change involves moving a plume from the second highest level to the highest level the costs of the sort trees change in a convex fashion. This is because as we enumerate the sort trees, endpoints which go through fewer passes have decreasing volumes resulting in monotonically decreasing cost savings while endpoints which go through more passes have larger and larger volumes resulting in monotonically increasing costs. Theorem 2 is another convexity

result. It shows that as we reduce a_h to a_h-1 then a_h-2, etc., $f_h(i,a_h)$, $f_h(i+1,a_h-1)$, $f_h(i+2,a_h-2)$ etc. form a convex function. Thus sequentially reducing the number of internal nodes at a given level (that is, moving a plume to a higher level) and reoptimizing the upper levels leads to a convex cost structure in the associated sort vectors. Theorems 1 and 2 provide the basis for step 2 of the algorithm. Theorem 3 is the basis for the detail of step 2 of the algorithm. It shows that, given the best solution so far in the enumeration, any better solution would involve moving some plumes to a higher level. Finally, if we have the minimum cost sort-tree with maximum allowable height H' and we wish to increase H' by one, Theorem 4 allows us to start the enumeration from the current optimal solution. This justifies step 4 of the algorithm.

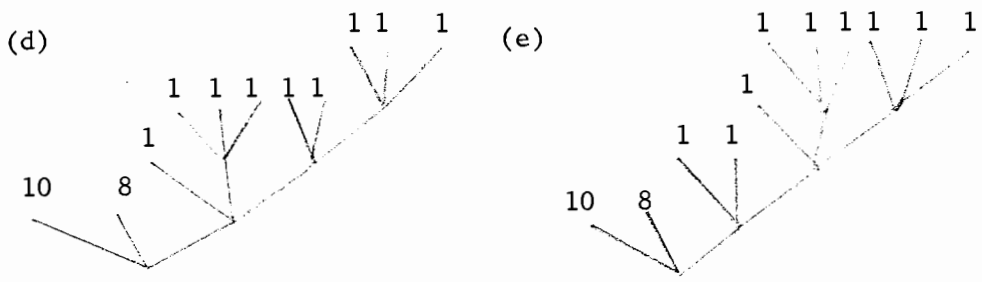
Figure 2 shows the sequence of sort-vectors evaluated by the algorithm for the problem with m=3, n=11, q=(10, 8, 1, 1, 1, 1, 1, 1, 1, 1, 1), $H \geq 4$, $c_h = h$, h=1,2,...,H. The figures above the endpoint nodes are the volume of checks at each endpoint. The optimal sort-vector is (1,1,3) with a cost of 45.

<u>$H'=3$</u>:



C(1,3,1)=57 C(1,2,2)=50 C(1,1,3)=45

<u>$H'=4$</u>:



C(1,1,2,1)=47 C(1,1,1,2)=49

FIGURE 2

In (a), (b) and (c) of Figure 2 we successively move plumes from level $H'-1$ to level H' using steps 2 and 3 of the algorithm. Since the costs are decreasing we stop only because it is not feasible to move more plumes up in the tree restricted to $H' = 3$ levels. In (d) of Figure (2) H' is increased by 1 using step 4 of the algorithm. From Theorem 4 it is not necessary to consider the sort vector (1,2,1,1)--we can always progress by moving plumes to higher levels in the tree. The sort-tree in (e) of Figure (2) is evaluated next but the costs increase so by the convexity of Theorem 2 the sort-tree in (d) is the best that can be obtained by shifting plumes from level three in the 4-level tree. Since there are no plumes to move from level 2 and the 3-level tree in (c) has a smaller cost we stop without examining any 5-level trees.

3. Computational Experience

The algorithm described in the previous section has been coded in FORTRAN and tested on a CDC 6400 computer. Four different runs were made for each setting of n (number of endpoints) and m (size of sorter). The endpoint volumes and sorting costs used for these experiments are shown in Table 1. Runs 1, 3, and 4 involved no restriction in level ($H = N$ where N is the number of internal nodes in the tree). Run 2 was the same as run 1 except that the maximum allowable number of levels of the tree was restricted to $H_1=5$, and run 3 was the same as run 4 (same random number seed) except that the costs, c_h were nonlinear.

TABLE 1

Description of Computer Runs

Run	Endpoint Volumes, q_i	Sorting Costs, c_h
1	$(n/2)^2$, $(n/2-1)^2$,...,2^2,1^2,1,1,...,1.	$c_h = h$, $h = 1,2,...$
2	$(n/2)^2$, $(n/2-1)^2$,...,2^2,1^2,1,1,...,1.	$c_h = h$, $1 \leq h \leq 5$
3	$q_i = r_i^2$ where r_i is a random number drawn from an exponential distribution with mean = 1000.	1.0, 2.5, 4.5, 7.0, 10.0, 13.5
4	(same as three above)	$c_h = h$, $h = 1,2,...$

Table 2 shows the optimal sort-vectors found by the algorithm together with the associated costs. In runs 1, 2, and 4 the total costs equal the total number of passes of checks through the sorter and the average number of passes per check varied from 1.94 to 2.66. The column headed "No. of trees" gives the number of candidate sort-vectors (trees) evaluated by the algorithm. These evaluations are carried-out efficiently by computing only the incremental changes in moving from one tree to the next. When n = 200 and m = 10, a complete enumeration of all trees restricted to five levels in height involves 1174 evaluations whereas the algorithm required only 39 iterations to find the optimal tree in run 2 of the experiment. Complete enumeration rapidly becomes impractical for larger values of H and N.

The number of trees evaluated by the algorithm depends on the relative sizes of the endpoint volumes. The least amount of computation occurs when the optimal sort-vector is the lexicographic maximum (minimum level tree). In this case the algorithm terminates after only five or six evaluations. The worst case for the algorithm is when the optimal tree has the maximum possible number of levels, N; however this case never occurs in the type of

TABLE 2

COMPUTATIONAL PERFORMANCE

Experiment				Optimal Solution			No. of Trees	Time (Seconds)
n	m	N	Run	Sort vector	Cost	Cost per check		
200	10	23	1	1,7,2,1,3,9	656,831	1.94	94	.40
			2*	1,7,2,2,11	656,861	1.94	39	.09
			3	1,7,9,4,1,1	385×10^6	2.49	46	.12
			4	1,6,9,4,2,1	747×10^6	1.94	51	.19
1000	10	111	1	1,10,30,10,3,2,8,47	111,135,503	2.66	744	3.62
			2*	1,10,31,15,54	111,236,512	2.66	140	.34
			3	1,10,43,35,15,4,2,1	$6,605 \times 10^6$	3.70	242	.71
			4	1,10,40,35,17,5,2,1	$4,616 \times 10^6$	2.59	345	1.40
2000	24	87	1	1,24,15,3,2,40,2	696,158,831	2.09	600	2.80
			2*	1,24,15,5,42	696,171,141	2.09	105	.31
			3	1,24,44,14,3,1	$10,561 \times 10^6$	2.81	70	.35
			4	1,23,42,16,4,1	$8,063 \times 10^6$	2.15	64	.37

*Restricted to five levels ($H_1 = 5$).

application considered here. For a given size sorter and similar distributions of endpoint volumes, the number of iterations required by the algorithm appears to increase approximately linearly with h in the experiments. As shown by run 3 the optimal sort-tree is quite sensitive to changes in the sorting costs. The computation times in the last column of Table 2 demonstrate that the algorithm is suitable for use in the real-time control of the check-sorting process as outlined in the next section of the paper. At present the algorithm is implemented as an interactive computer program which allows the system designer to rapidly assess the affects of various factors such as changes in the costs, c_h, different

sorter sizes, and the separation of different 'special' end-points on the first or subsequent passes.

The data used in the experiments described above were contrived to test the computational performance of the algorithm. Detailed endpoint volume data are not presently available from the banks consulted during this research. However, it has been possible to carry out one experiment using available data collected over a one hour period from an actual sorting process. In this case the sorter had 12 (non-reject) pockets, there were 265 endpoints and the sort tree involved a maximum of 4 levels. The data represented approximately 160,000 checks with a total value of $50 million. To compensate for 'special' pockets as described earlier, the algorithm was run for this endpoint data with m=10. The results given in Table 3 show a 5% decrease in average number of passes per check. This represents a substantial saving since on a pro-rata basis, an additional $2,500,000 worth of checks might have been processed in the same time period if the optimal sort-tree had been used.

TABLE 3

COMPARISON OF ACTUAL AND OPTIMAL SORT-TREES

	Sort-vector a	Average No. of Passes per Check
Actual sort-tree	1,6,13,4	1.96
Optimal sort-tree (H = 4)	1,7,16,6	1.87

4. Sorting Patterns When Time Constraints are Present

The sort tree obtained from the algorithm presented in the previous section can be used without modification to achieve efficient processing of the checks associated with the bank's own customers. To implement the processing procedure it

makes sense in this case simply to sequence the set-ups in left-to-right order on each level and to complete all the set-ups on any one level of the tree before proceeding to the next. This will be optimal if there is a positive probability of a system break-down since the end-points with large volumes are processed first.

In processing the checks drawn on other banks various groups of end-points have to be processed before different deadlines on each day so that the processing bank can receive credit for the dollar value involved. The bank essentially loses one day's interest on the value of all checks which miss their deadlines. In this section we suggest a procedure for modifying the sort-tree and determining a sequence of processing steps (visits to internal nodes). This sequence allows for the presence of deadlines and the dollar value of the checks and yet still maintains the minimum average number of passes per check. The combination of the sort-tree and the specified sequence of processing steps is called a "sort-pattern." We assume a real-time processing environment in which it is possible to determine a separate sort-pattern either for each incoming batch of checks or for all batches of checks for which processing is begun during a specified time interval. The sort-pattern will depend on the time at which processing starts both because of the deadlines and because the expected end-point volumes and dollar values vary throughout the day. To determine the sort-pattern for a batch of checks the following steps are proposed:

(1) Given the time of day and information concerning the origin of the checks the expected distributions of check-volumes and values by end-point is found from a table of stored values.

(2) The algorithm described in the previous sections is run and a sort-tree which minimizes the total expected sorting time for the batch is determined. At this point the end-points going from left to right in the sort-tree are arranged in decreasing order by volume.

(3) The end-points at each level in the sort-tree determined during step(2)are ordered from left-to-right by increasing time to deadline and within the group of end-points for each dead-line by decreasing dollar value. The resulting sort-tree may appear as in Figure 3(a) where the letters appearing above the external nodes refer to the associated deadline with deadline "a" being the most imminent and deadline "b" the next most imminent and so on.

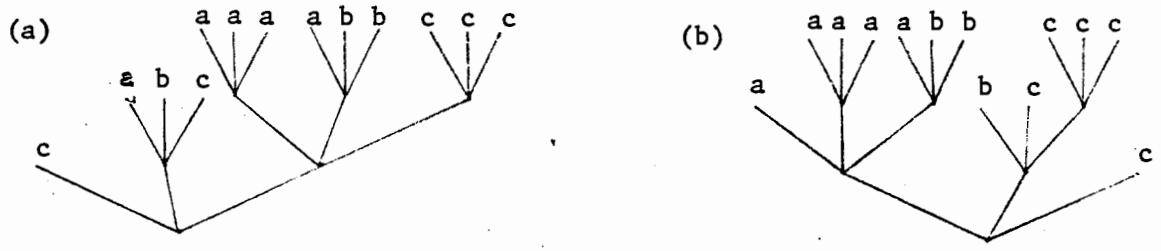


FIGURE 3

(4) A modified sort-tree (see Figure 3(b))is now obtained by associating each internal node with the highest priority deadline in the subtree for which it is a root and moving it to the left until an end-point with the same or higher priority is encountered.

As the modified sort-tree involves the same number of set-ups as the sort-tree determined in step (2) and each end-point goes through the same number of passes, it will also involve the minimum expected total processing time. However the new sort-tree has the advantage that the number of setups required to completely process all the end-points associated with the earliest dead-line,

"a", is minimized within the class of optimal sort tries. Also, going from left-to-right within each level, the internal nodes are associated with dead lines of decreasing urgency and within the group of internal nodes associated with each dead-line the expected dollar values associated with the endpoints are decreasing.

(5) To complete the specification of the sort-pattern for the given batch (or group of batches) it is now necessary to determine the actual time sequence of set-ups (visits to internal nodes) which is to be employed. For this purpose a job-shop scheduling procedure, [6], can be employed to determine the sequence which minimizes the expected value of the checks which miss dead-lines.

APPENDIX

The algorithm presented in Section 2 is based on the following theorems:

Theorem 1: Let $C(a_1,a_2,\ldots,a_h)$ be the cost of the sort-tree represented by $a_1,a_2,\ldots,a_h$. $C(a_1,a_2,\ldots,a_{h-1}-k,a_h+k)$ is a convex function over the feasible range of k.

Proof: As k increases plumes with non-decreasing volume endpoints are moved from level h-1 to level h and endpoints with non-increasing volumes are moved to level h-2. With costs non-decreasing and savings non-increasing the theorem holds.

Theorem 2: In the dynamic program introduced previously:

$$(3)\qquad f_h(i+1,a_h-1) + (c_h - c_{h-1})q_i - f_h(i,a_h)$$
$$\leq f_h(i+2,a_h-2) + (c_h - c_{h-1})q_{i+1} - f_h(i+1,a_h-1)$$

and

$$(4)\qquad f_h(i-m+1,a_h) - f_h(i,a_h) \leq f_h(i-2m+2,a_h) - f_h(i-m+1,a_h)$$

Proof: To prove (4) we proceed by induction on h starting at h_1 and working down. Let h' be the highest level in the tree with $a_{h'+1} < ma_{h'}$. For $h = h'-1$, (4) implies that we are bringing a succession of plumes from some level less than h'-1 and placing them at level h'. As in Theorem 1, since the endpoints for each added plume are non-decreasing in volume, (4) holds for $h = h'-1$.

Suppose (4) is true for $h+1,h+2,\ldots,h'$. We have:[1]

$$(5)\qquad f_h(i,a_h) = \min_{j\geq i} \{f_{h+1}(j,ma_h - j+i) + (c_h - c_{h-1}) \sum_{k=i}^{n} q_k\}$$

Modifying our indexing:

---

[1]To simplify the notation and without loss of generality we assume $j_1(i) = i$

$$(6)\quad f_h(i-m+1,a_h) = \min_{j-m+1\geq i-m+1} \left\{ f_{h+1}(j-m+1, ma_h - j+i) + (c_h - c_{h-1}) \sum_{k=i-m+1}^{n} q_k \right\}$$

$$(7)\quad f(i-2m+2,a_h) \geq \min_{j-2m+2\geq i-2m+2} \left\{ f_{h+1}(j-2m+2, ma_h - j+i) + (c_h - c_{h-1}) \sum_{k=i-2m+2}^{n} q_k \right\}$$

The upper limit of the range for j in (5), (6) and (7) is decreasing from feasibility considerations since there are more banks at higher levels in the tree. Therefore, looking at the individual terms in (5), (6) and (7), (4) follows from the induction hypothesis and the monotonicity of the q_k's.

In (3) we are moving plumes from level h to some higher level in the tree. To prove (3), we have (5) as before together with:

$$(8)\quad (c_h - c_{h-1})q_i + f_h(i+1, a_h - 1) = \min_{j-m+1\geq i+1} \left\{ f_{h+1}(j-m+1, ma_h - j+i) + (c_h - c_{h-1}) \sum_{k=i}^{n} q_k \right\}$$

$$(9)\quad (c_h - c_{h-1})q_{i+1} + f_h(i+2, a_h - 2) = \min_{j-2m+2\geq i+2} \left\{ f_{h+1}(j-2m+2, ma_h - j+i) + (c_h - c_{h-1}) \sum_{k=i+1}^{n} q_k \right\}$$

As above, the upper limit of the range for j in (5), (8) and (9) is decreasing from feasibility considerations. Since the number of roots available in (5), (8) and (9) is the same (equal to $ma_h - j+i$) we may use inequality (4) which gives the result.

Theorem 3: Let:

$$(10)\qquad a_1, a_2, \ldots, a_{h-1}, a_h, \ldots, a_H$$

represent a sort-tree where $a_h, a_{h+1}, \ldots, a_H$ provides the minimum cost sort-tree for $a_1, a_2, \ldots, a_{h-1}$ fixed. Assume a plume is relocated so that:

$$(11)\qquad b_1, b_2, \ldots, b_{h-1}, a_h + 1, a_{h+1}, \ldots, a_H$$

is a sort-tree with $b_i \leq a_i$ for $i = 1,2,\ldots,h-1$. Given $b_1, b_2, \ldots, b_{h-1}$, let

(12) $b_1, b_2, \ldots, b_{h-1}, e_h, e_{h+1}, \ldots, e_H$

be the minimum cost sort-tree with $e_h \geq a_h + 1$, then $e_h = a_h + 1$ and $e_r = a_r$ for $r = h+1, h+2, \ldots, H$.

Proof: The endpoints that move to a higher level when the number of internal nodes at level h is increased from a_h to a_h+1 have volumes at least as great as those already at this level. Now, applying the convexity result (3) to sort-vector (12) at level h, we see that the theorem holds.

Theorems 1, 2 and 3 complete our characterization of restricted level trees. We next characterize the effect of increasing the number of levels on the costs of a shifting a plume. A superscript is added to f_h as defined in (2) to indicate the height restriction.

Theorem 4: In the dynamic program introduced previously:

$$(13) \qquad f_h^H(i+1,a_h-1) - f_h^H(i,a_h) \geq f_h^{H+1}(i+1,a_h-1) - f_h^{H+1}(i,a_h)$$

$$(14) \qquad f_h^H(i-m+1,a_h) - f_h^H(i,a_h) \geq f_h^{H+1}(i-m+1,a_h) - f_h^{H+1}(i,a_h)$$

This theorem shows that the increase in cost incurred by adding a plume or removing an internal node from a given level is not greater when the extra level is available. Therefore, every shift of a plume which occurs with H levels, would also occur with $H+1$ levels in the restricted height tree. As a consequence, the optimal sort tree restricted to at most $H+1$ levels can be constructed by starting with the optimal sort tree with H levels and shifting plumes to higher levels using the restricted level algorithm from the previous section. Clearly, once the number of levels has increased to the point where the highest level contains no plumes, we may stop.

Proof of Theorem 4: Our proof is by induction. Let $a_1,\ldots,a_H$ represent a sort-tree with a maximum height of H. Adding another level and optimizing between levels H and H+1 gives an optimal sort tree $a_1,\ldots,a_{H-1},b_h,b_{H+1}$. Since $b_H \leq a_H$, going from $a_1,\ldots,a_{H-1},a_H$ to $a_1,\ldots,a_{H-1}-1,a_H+1$ increases the number of passes for endpoints of higher volume than going from $a_1,\ldots,a_{H-1},b_H\ b_{H+1}$ to $a_1,\ldots,a_{H-1},b_H+1,b_{H+1}$. In addition, there can be a cost saving reoptimization between b_H+1 and b_{H+1}. Therefore, at level h = H-1 (13) holds and in like manner (14) holds.

Assume (13) and (14) hold for h+1,...,H and our optimal sort vectors for $a_1,\ldots,a_h$ fixed are $a_1,\ldots,a_h,b_{h+1},\ldots,b_{H+1}$ and $a_1,\ldots,a_h,a_{h+1},\ldots,a_H$. By the induction hypothesis $b_{h+1} \leq a_{h+1}$. We now prove (13). There are several cases which must be considered. The first case is when $a_{h+1} < m(a_h-1)$. Here we remove one plume from level h and add it to level h+1 and reoptimize from level h+1. The first possibility is that $a_{h+1} = b_{h+1}$. If that is the case, the plume is added to level h+1, the same endpoints are moved to level h+1 at the same costs in both trees. The next step is to reoptimize at level h+1 by considering $f^H_{h+1}(j-m+1,a_{h+1}+1)$, $f^H_{h+1}(j-m+2,a_{h+1})$, etc., and $f^{H+1}_{h+1}(j-m+1,a_{h+1}+1)$, $f^{H+1}_{h+1}(j-m+2,a_{h+1})$ etc. By the induction hypothesis the savings at each iteration (if any) are greater in the H+1 level tree and the savings may continue for more iterations in the H+1 level trees therefore, (13) holds in this case.

The second possibility with $a_{h+1} < m(a_{h-1})$ is when $a_{h+1} > b_{h+1}$.

Let $p_i,\ldots,p_j=h$ and $\hat{p}_i,\ldots,\hat{p}_{j'}=h$ in the H level tree and H+1 level tree respectively. By the optimality of $a_h,\ldots,a_H$ given $a_1,\ldots,a_{h-1}$,

$$\gamma = f^H_{h+1}(j+2,a_{h+1}-1) - f^H_{h+1}(j+1,a_{h+1}) \geq (c_{h+1} - c_h)\, q_{j+1} \geq (c_{h+1} - c_h)\, q_{j'} . \tag{15}$$

The cost of moving a plume from level h to $h+1$ in the H level tree is $(c_{h+1} - c_h)(q_{j-m+1} + \ldots + q_j)$ and in the $H+1$ level tree is either $(c_{h+1} - c_h)(q_{j'-m+1} + \ldots + q_j + \ldots + q_{j'})$ if $j \geq j'-m+1$ ('overlapping' case) or $(c_{h+1} - c_h)(q_{j'-m+1} + \ldots + q_{j'})$ if $j \leq j'-m+1$ (non-overlapping case). Consider a move from state (i,a_h) to $(i+1,a_h-1)$ in the H-level tree. If in reoptimizing at level $h+1$ there are any changes in the H level tree, a cost at least as great as γ is incurred each time p_{j-m}, p_{j-m+1} etc. are reduced from $h+1$ to h. That is, at least γ is traded for $(c_h - c_{h-1})q_{j-m}$. By (15) the cost of reducing a nonoverlapping endpoint to its former level in the H-level tree is greater than the cost of initially raising the level of a non-overlapping endpoint in the $H+1$ level tree. Therefore, a cost of at least γ is incurred for each endpoint raised while moving the plume in the H level tree. If there is no overlap (13) clearly holds. If there is overlap, optimize at level $h+1$ in the H level tree. If no overlapping endpoints are reduced (13) holds, again because of (15). If, in the H level tree, overlapping endpoints are reduced, at some point the number of internal nodes in the H level tree is equal to $b_{h+1}+1$. Applying the induction hypothesis as in the first case proved, (13) again holds.

Now we consider the case where $a_{h+1} \geq m(a_h - 1)$. Here the reduction of a_h by one involves the displacement of more than one plume. Since $b_{h+1} \leq a_{h+1}$, at least as many plumes are displaced before optimization in the higher levels in the H level tree than in the $H+1$ level tree incurring at least as great a cost. We need only iterate at level and by the induction hypotheses for (13) and (14), (13) holds. Also, (14) follows directly from (13) at level h.

References

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

2. 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, 1972.

3. Huffman, D.A., "A Method for the Construction of Minimum - Redundancy Codes," Proc. IRE 40, 1098-1101 (1952).

4. Knuth, D.E., The Art of Computer Programming, Vol. 1, Addison-Wesley, Reading, Mass., 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 Mathematical Programming Approach to the Scheduling of Check Sorting Operations," Discussion Paper No. 164, Center for Mathematical Studies in Economics and Management Science, Northwestern University, Evanston, Illinois.

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