[illegible]
[illegible]
DRC-70-09-80

A FORWARD SIMPLEX METHOD

by

Jay E. Aronson, Thomas E. Morton & Gerald L. Thompson

DRC-70-09-80

April 1980

Authors' address:

Graduate School of Industrial Administration
Carnegie-Mellon University
Pittsburgh, PA 15213

Prepared under National Science Foundation Grant ENG 7725691.

A Forward Simplex Method

Jay E. Aronson
Thomas E. Morton
Gerald L. Thompson

ABSTRACT

Forward algorithms are an approach to solving dynamic problems by solving successively longer finite horizon subproblems, terminating when a stopping rule can be invoked (or planning horizon found). Such procedures are now available for a large number of special structure models. Here we discuss the development, implementation and testing of such a procedure for general dynamic (staircase structure) linear programs. Tests suggest that solution time is linear in the problem length, versus quadratic or cubic for earlier linear programming codes. Planning horizons are often obtained.

1. Introduction

Forward algorithms are an approach to solving dynamic problems by solving successively longer finite horizon subproblems, terminating when a stopping rule can be invoked (or planning horizon found). (See [4], [14])

Formally, a forward algorithm is a procedure which solves first a one period problem, then a two period problem, then a three period problem, and so on. The solution to the T period subproblem is ideally found by augmenting the T-1 period optimal solution. Such a forward algorithm is termed efficient. A forward algorithm thus develops a sequence of better approximations to an optimal first decision (which is often all that is desired). When the current approximation of the first period decision is guaranteed to be "good enough", then computation stops. A forecast horizon is the number of periods of forecast information required to guarantee that the first several periods are optimal irrespective of information past it. The initial periods are called the planning horizon corresponding to the forecast horizon. Heuristic planning horizons may also be considered. These horizons may be defined for when the decisions for any longer finite subproblem have stopped fluctuating very much (apparent planning horizon); are guaranteed to be exactly optimal (exact planning horizon); guaranteed to be within ϵ of cost of the optimum (near cost horizon); or guaranteed to be within ϵ of the optimal decision (near policy planning horizon). Forecast horizons are also described as being apparent, exact, near cost or near policy. A planning horizon procedure is an efficient stopping rule for a forward algorithm, since it determines the longest finite subproblem that must be solved. A planning horizon decomposes a dynamic problem into three parts: the "stable" part up to and including the planning horizon, the second part from just beyond the planning horizon to the forecast horizon, and the third part past the forecast horizon- The values of the dynamic variables at the planning horizon become the initial conditions for the second part of the problem. Forward algorithms and planning horizon procedures are now available for a large number of special structure models ([4], [U], [12], [17], [13]). For the first time the authors have extended these techniques to general dynamic linear programming problems.

We now torn our attention to planning horizons relative to such staircase structure linear programs. Basic solutions to staircase structure linear programs typically have the property that similar types of activities persist In the basis over several consecutive time periods (Perold and Dantzig [161). In [1], we observed that these activities isolate blocks of variables by time periods* These blocks form a natural decomposition, which in a sense, is equivalent to a planning horizon. For those models which have theoretical planning horizons, the natural decomposition occurs at such horizons. For models with no known theoretical horizon, the natural decomposition of problems can sometimes still be utilized to increase the efficiency of the simplex method.

In this paper we discuss the development, Implementation, and testing of the Forward Simplex Method, which is a forward algorithm for solving general staircase structure programs. Problem size has been found to be a major bottleneck in large-scale mathematical programming. By exploiting the natural decomposition of the staircase linear program, the forward simplex method solves very large problems in core, since the size of the working tableau (basis) is reduced. In the Implementation stage, we even discard part of the updated tableau after it is no longer needed, retaining only enough of it to continue the procedure. Furthermore, under certain conditions, problems having arbitrarily large numbers of periods are solvable.

In section 2 we present the general dynamic linear programming model. Section 3 gives a brief exposition on planning horizons. In section 4, we discuss the Forward Simplex Method. A stannary of the computational study of Aronson [2] appears in section 5. Tests suggest that the solution time is linear.in the problem length, versus quadratic or cubic for earlier linear programming codes. Planning horizons are often obtained. In appendix A we discuss specific programming techniques for implementing the algorithm. These include the issues of basis or tableau representation, augmentation, pivoting, primal feasibility following augmentation, horizon detection, etc. Appendix B gives improvements for later versions of the code* t

2. The Modal

Consider Che general staircase linear program:

(a) $\text{ain} \sum_{f1}^{T} c_c T_c .$

subject to

tt*(b) $A_1 X_1 . . -d_1$

(c) $B_{t-1} X_{t-1} + A_t X_t = d_t, \quad t-2,...,T$

(<*) $X_c \geq 0, \quad t*l,...,T$

where c_t is 1 by n_t, A_t is m_t by n_t, B_t is BL_{t+1}* by n_t, d_t is m_t by 1, and X_t is n_t by 1. We assume, for simplicity, A_t « A and B_t *B, for t*l,...,T, where A and B are fixed matrices. This assumption can easily be relaxed. The staircase structure is evident in Figure 1.

In staircase models, the vector X_t is divided into two sets: local and pass-on variables. A pass-on variable is one which has a non-zero column in B. That is, it appears in the constraint set of the next time period, thus directly influencing it. Usually there are only a few pass-on variables as compared to the total number of variables. In the dynamic programming and control theory approaches to these problems, the state variables are the pass-ens- For example, in a production smoothing model, the ^n<ifpg inventory and work force level are pass-on variables. A local variable has a zero column in B. It does not directly link to the next period, but indirectly affects them through the pass-cms. Hiring of workers is an example of a local variable. Its value directly affects the current workforce level (a pass-on). The linkage to the next period is through the workforce level. Furthermore, there may be local constraints (zero rows of B) only on the local variables; and only on the pass-on variables. These

may be upper bounds on the variables, etc. Pass-in constraints are Che nonzero rows of B. These are the constraints that pass the pass-on variables into the next period. Thus we partition A and B as shown in Figure 2.

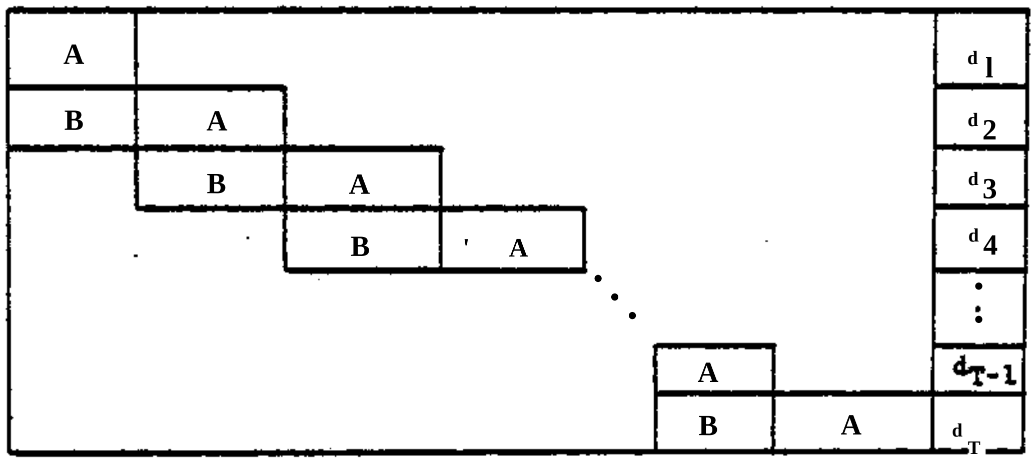


Figure 1: Block structure diagram of the constraint matrix of die general staircase linear program.

$$\begin{array}{cc|c|c|}
 & & \text{Columns} & \\
 & & \text{local} & \text{pass-on} \\
\hline
 & \text{local} & A^1 & 0 \\
\hline
\text{Sows} & \text{pass-in} & A^2 & A^3 \\
\hline
 & \text{local} & 0 & A^4 \\
\hline
\end{array}
\qquad\qquad
\begin{array}{|c|c|cc}
 \text{Columns} & & & \\
 \text{local} & \text{pass-on} & & \\
\hline
 0 & 0 & \text{local} & \\
\hline
 0 & B^1 & \text{pass-in} & \text{Rows} \\
\hline
 0 & 0 & \text{local} & \\
\hline
\end{array}$$

A B

Figure 2: Partitioning of the A and B matrices for the general staircase linear program.

3. Planning Horizons and Natural Decomposition

Staircase linear programming problems often result from planning models• In many planning situations, a decision must usually be made only for the first: period• Accurate data are typically available for early time periods, but equally accurate forecasts far into the future are impossible to obtain. So, once a first period decision has a low probability of changing, the decision can be made. As more reliable data become available, the problem can be re-solved in the next period. This technique is sometimes called the "rolling horizon" method [14].

A forward algorithm is a foxaal procedure which solves longer and longer finite horizon subproblems. Ideally it augments the (T-l)-period subproblem optimum to generate the solution to the T-period subproblem. Consider (1) to be a T-period subproblem of a longer staircase linear program of length, say T_p. When a forward algorithm is applied to (1), the staircase structure is maintained. The new ''steps" of the staircase usually consist of blocks larger than the original A matrix. See Figure 3a. Specifically, we define a block B to be a maximal rectangular submatrix such that if a simplex pivot is chosen on any entry in the block, then the pivot will not affect any entry in any other block. We define the set of affected elements, S(B), to be the submatrix that contains all rows and columns affected by pivots chosen in block B. We define an extended block, E(B), to be the maximal rectangular submatrix contained in S(B). Clearly B C E (B) £ S (B) . We say that two blocks B_i and B_j, (and also the extended blocks $E(B_i)$ and $E(B_j)$) are adjacent if $S(B_i)$ *d* $S(B_j)$ * $. We define a pinch block, p_{jL}, to be the rectangular submatrix formed by the intersection of two adjacent extended blocks; P_i - *Z(B±)* H E ($^B{}_{i+}$i) - A *P^ck* block is the connecting link between adjacent extended blocks. Usually a pinch block has smaller dimensions than A, because there are fewer pass-on variables than locals. The initial tableau has only linking in the columns, thus the pinch blocks are empty matrices since the adjacent extended blocks do not overlap in the rows. As pass-on variables enter the basis, the rows overlap, forming non empty pinch blocks. For example, in the Wagner-Whitin model [20], inventory is the only pass-on variable. When it is basic, it links the rows, but when it is non basic an empty pinch block occurs, even though its column still links two adjacent #xt?eiyJ4Ki blocks*

The pinch blocks form a natural decomposition of (1) • They isolate adjacent blocks from the effects of pivoting by at least one pivot* If a pivot is performed on an entry in a block, then only entries in its extended block *ax^* affected. Even if a pivot is performed on an entry in a pinch block, only the two adjacent extended blocks are affected. Usually they will form one larger extended block. Extended blocks adjacent to this new larger one are not affected by the pivot* For example, in Figure 3, a pivot on an entry in block B_3 can not affect entries in blocks B_1 and B_2- A pivot on an entry in extended block $E(B_3)$ affects entries in block B_3; may affect those in B_2; but never entries in B_1. Thus B_1 and B_3 are isolated from each other by at least one simplex pivot.

For certain models, the natural decomposition caused by pinch blocks occur where there are exact planning horizons [1], [4], [11], [12], [13], [17], [19]. For these models, once such a horizon is found, computations may stop. For the special model discussed in [1] and [4], exact planning horizons can be identified by a planning horizon procedure, by dual analysis, or by the adjoint analysis of optimal control theory.

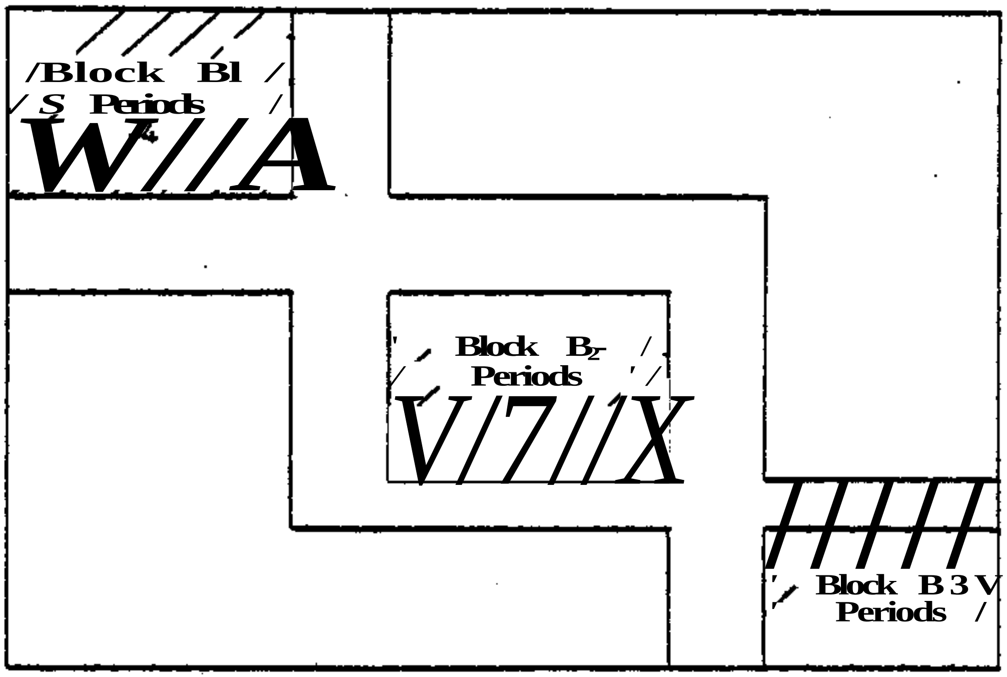


Figure 3a: Possible natural decomposition of a 7 period staircase linear programming problem* Basic and non basic variables only from periods 1-4 appear in block Bi, 4-6 in B2, «** 6-7 in B3- Pivots on entries within a block cannot affect entries in any other block.

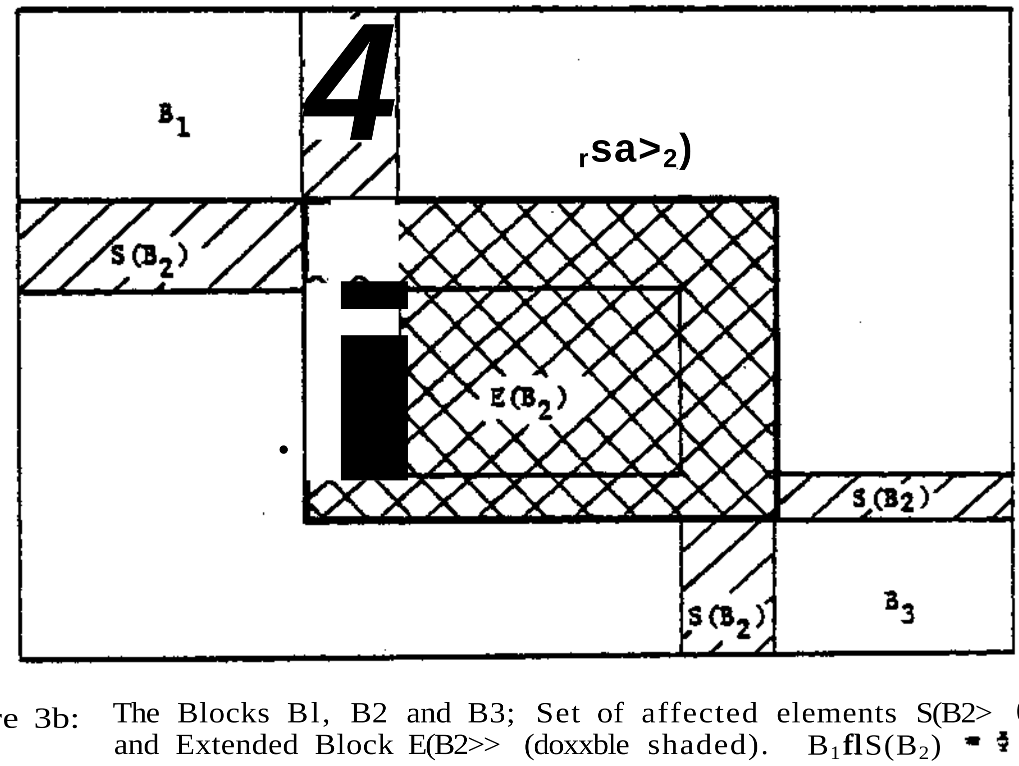


Figure 3b: The Blocks Bl, B2 and B3; Set of affected elements S(B2> (shaded); and Extended Block E(B2>> (doxxble shaded). $B_1 \cap S(B_2) = \emptyset$ and $B_3 \cap S(B_2) = \emptyset$.

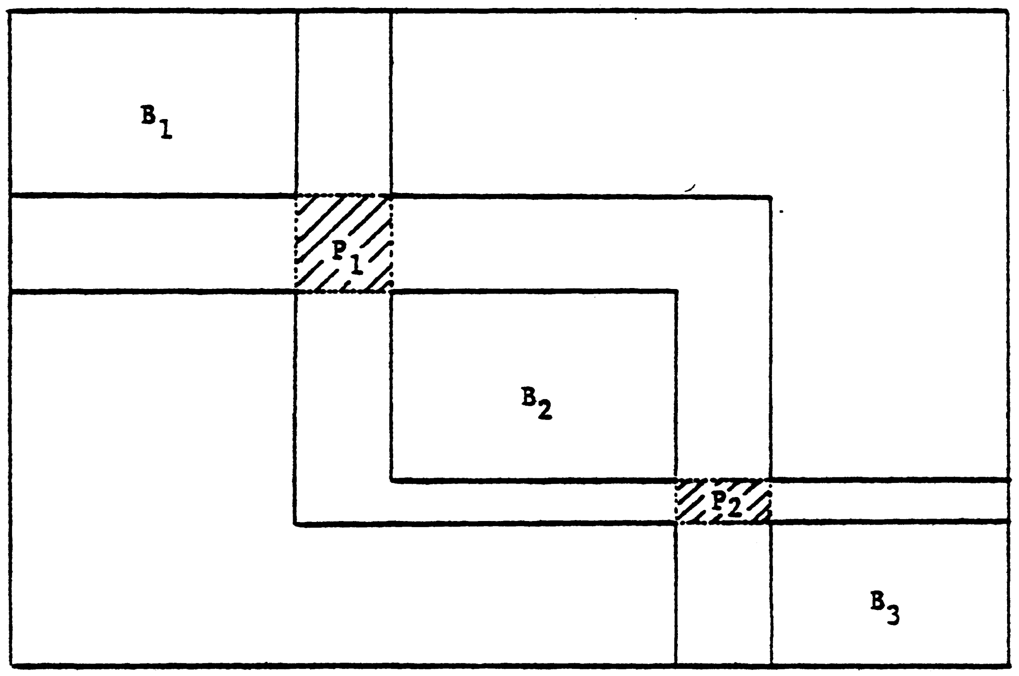


Figure 3c: Pinch blocks P_1, P_2 in the tableau. The pinch blocks isolate pivoting effects from one block to the next.

For Che general dynamic linear programing problem, it may noc be possible to find an exact planning horizon. Each problem has its own characteristics and the existance of exact horizons depend on the specific structure of the model. However, the natural decomposition caused by tmxltiple pinch blocks can be used to define a heuristic planning horizon for general staircase linear programs. As longer finite subproblems are solved, multiple pinch blocks tend to prevent variables in the earliest blocks from changing value or from entering or leaving the basis. So, the probability that variables in the first block change value is "low enough" for- any longer finite subproblem. A reasonable stopping rule for a forward linear programing algorithm is when a predetermined number of pinch blocks occur in the tableau.

We term the length of the subproblem (1) a heuristic forecast horizon when the stopping rule is invoked. By specifying the number of pinch blocks in advance and not the forecast horizon, we define a true forward algorithm and planning horizon procedure [14]•

In a typical planning situation, multiple pinch blocks can be expected. For these problems, the stopping rule implies finlteness. The number of pinch blocks to use must be determined by the problem solver on the basis of his knowledge of the model. The forward simplex method discussed in the next section exploits these heuristic horizons to devise an efficient algorithm for solving the general staircase linear programming problem.

4. The Forw△Td Si^rol^x Method

The forward simplex method is a forward algorithm applied to staircase linear programs (1). Let (1) be a T-period subproblem of some longer problem with length I_p. The forward simplex method partially solves the 1-period subproblem, augments this solution to form an initial basic feasible solution to the 2-period subproblem, solves this one, and so on for T • 3 to T_p. When T • T_P the whole problem is solved. It is efficient because it exploits a natural decomposition of the problem. No reordering of the rows and colinns are necessary to maintain the staircase structure. Thus there is an automatic spike reduction [7], [3]. Blocks appear in the tableau as shown in Figure 3. Pivoting computation in later periods usually affect only primal and dual variables within later blocks.

Intuitively, later period forecasts should not have much impact on early decisions. The natural decomposition of the staircase problem tends to isolate early decisions from the effects of later periods.

Let the columns of A and B be partitioned as explained in section 2. The maximum tableau dimensions are m by a. The A and B matrices have much smaller dimensions. Let *S*$_T^*$ be the "optimal" basic solution to (1), with the pass-on variables in period T restricted to be non basic at zero[1], unless T *m* I_P.

We start by solving die 1-period subproblem. We set T to 1 and load A into the upper left corner of the tableau. A slack starting basis is used. We optimize to find S_1^*, restricting the pass-on variables to be non basic at their lower bound zero. The initial candidate list for basis entry is comprised of only the local columns; 1 to the rightmost local

[1]The pass-on variables are restricted to be non basic at zero to prevent excessive bookkeeping and die need for dual simplex pivots to attain primal feasibility following augmentation.

column. The reduced costs of the candidate list are scanned from right to left, the first positive reduced cost identifies the entering variable. The minimum ratio test is performed from bottom to top. For the 1-period sttbproblem the candidate list size does not change.

Once S_1^* is attained, we augment it to form an initial basic feasible solution to the 2-period subproblea. In general, once S_T^* is found, we increment T by 1 aod form an initial basic feasible solution S_T^1 from S_{T-1}^*. fte now augment the tableau to include the rows and columns associated with the period (T-l) B matrix and the period T A matrix. We copy the B matrix directly below the position of the previous period's A matrix* We copy a new A matrix Into the tableau just to the right of this B matrix. The right hand side and costs for period T are copied in as well. The initial basic feasible solution is the optimal basis to die (T-l)-period subproblem plus the slacks in period T (some may be artificial). The initial candidate list following augmentation consists of the pas3-on columns in period T-l, and the period T locals. It is clear that immediately following augmentation columns not in the list are not eligible for basis entry, else the conditions for S_{T-1}^* «ould be violated, a contradiction. As before, the current period pass-on variables remain non basic at zero. During the optimization of this subproblem, the candidate is expanded (to the left) to include those columns up to the earliest one affected by the current series of pivots.

If a pivot entry is in a block, it affects only those entries in the extended block. If the entry is in the extended block, but not in a pinch point, it may affect only entries of its extended block and a part of the adjacent extended block not containing the adjacent block itself.

If the entry is in a pinch block, the pivot may affect entries in both the adjacent extended blocks (often forming one larger extended block).

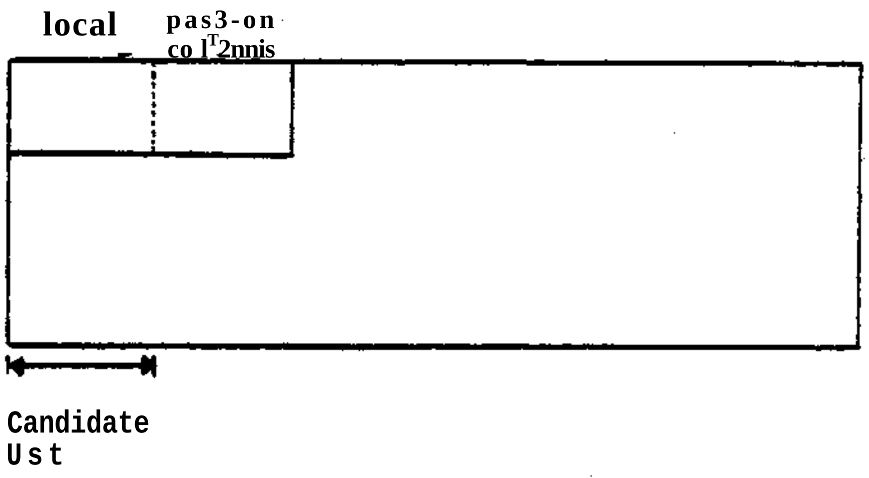
Ideally the pinch blocks prevent pivoting activity in later periods from affecting early period variables. In fact if a pinch block contains only non positive entries, its entries cannot be chosen as pivot elements.

We index the first aod last non zero entry of each row and column to define the extended blocks and reduce pivoting work. Following each pivot, these indices are updated, as well as the list of columns included in the candidate list. Pivoting continues until S_T^* is attained. We use the standard tests for infeasibility and unboundedness. The notions of pivoting and augmentation are shown in Figure 4.

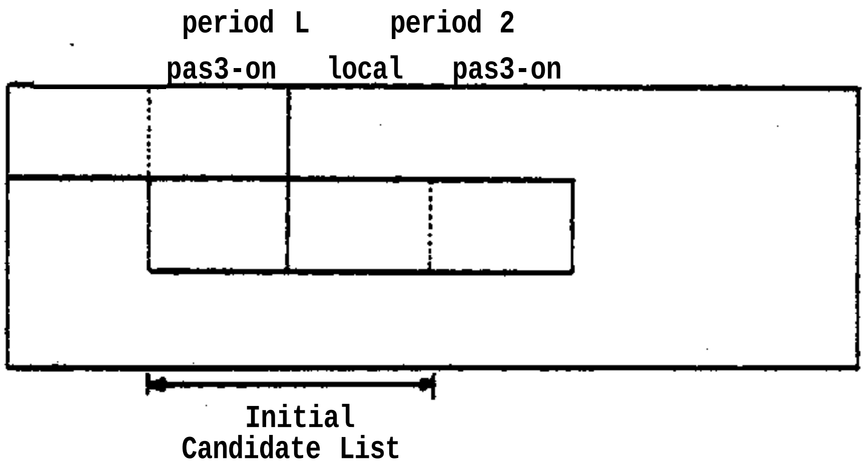
We continue augmenting and pivoting until either the T_P period problem is solved or the entire allowable tableau fills up. When the tableau is full, we consider the available tableau space to be a small window into the entire problem. We slide this tableau window down and to the right, discarding early stable data from core, and augmenting the newest data into the tableau window. Instead of actually sliding the window, we employ a wrap-around tableau, which puts tableau entries appearing below and to the right of the window in the upper left hand part of the tableau window. An illustration of this appears in Figures 5 and 6, and will be explained more fully next.

To detect a heuristic horizon we scan the tableau for a pinch block which allows enough space for at least one augmentation as in Figure 5. We discard the stable block from core, zero that portion of the tableau, and augment as in Figure 6. We define the pointers FROW and FCOL to be the first valid row and column of the tableau, LASKOW and LASCOL to be the last, and BORROW and EORCOL as lower bounds on the row and column pivoting

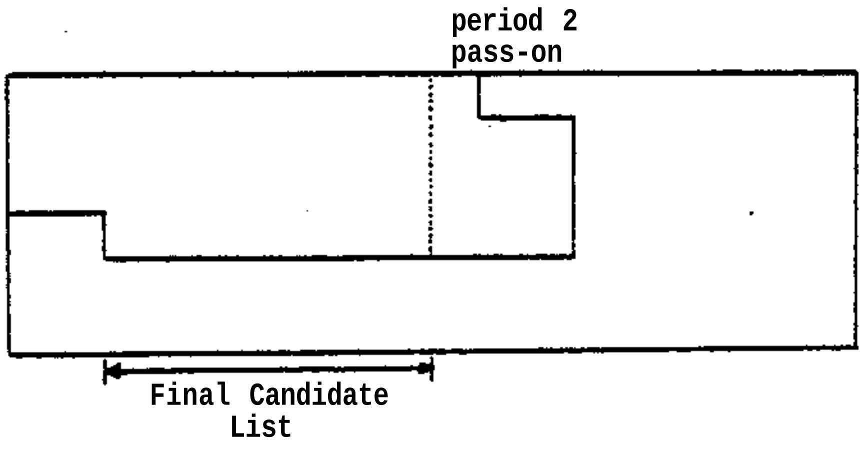
Figure 4: Diagramatxc representation of the tableau* for the 1, 2 and 3-period subprobiems of a three period dynamic linear program.



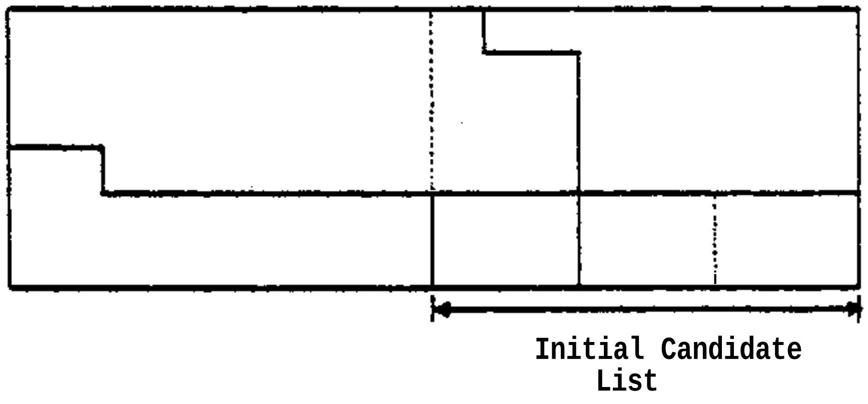
(4a) 1-period subproblem, optimal tableau



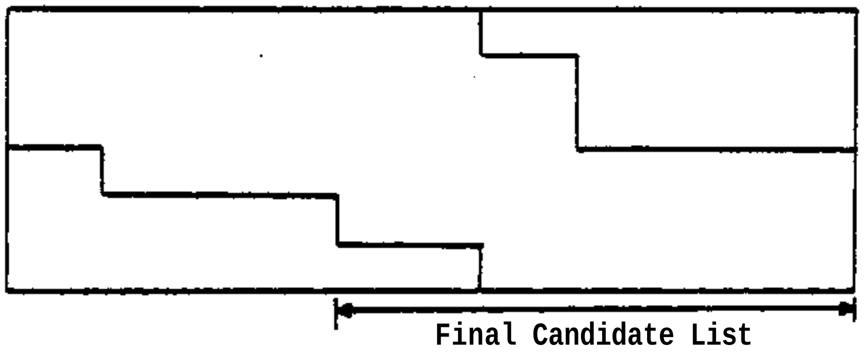
(4b) 2-period siibproblem, initial tableau following augmentation



(4c) 2-period subproblem, optimal tableau



(4d) 3-period subproblem, initial tableau following augmentation



(4e) 3-period subproblem, optimal tableau

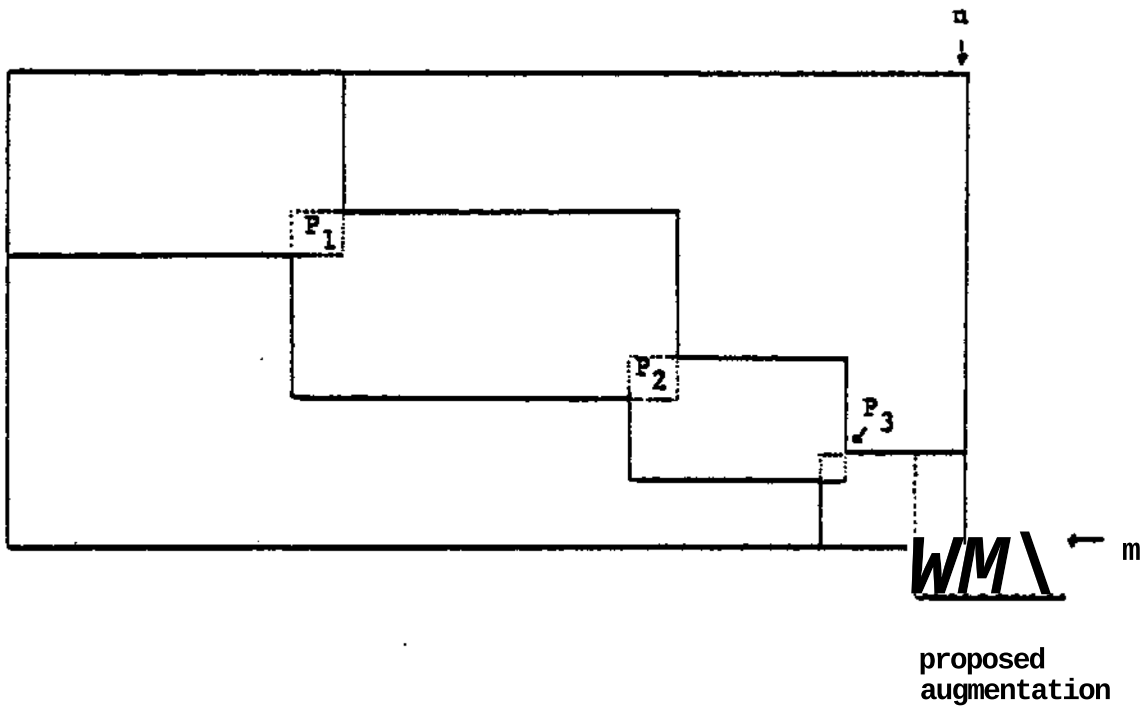


Figure 5: A full tableau window with the proposed augmentation. The pinch point P_1 indicates the heuristic planning horizon.

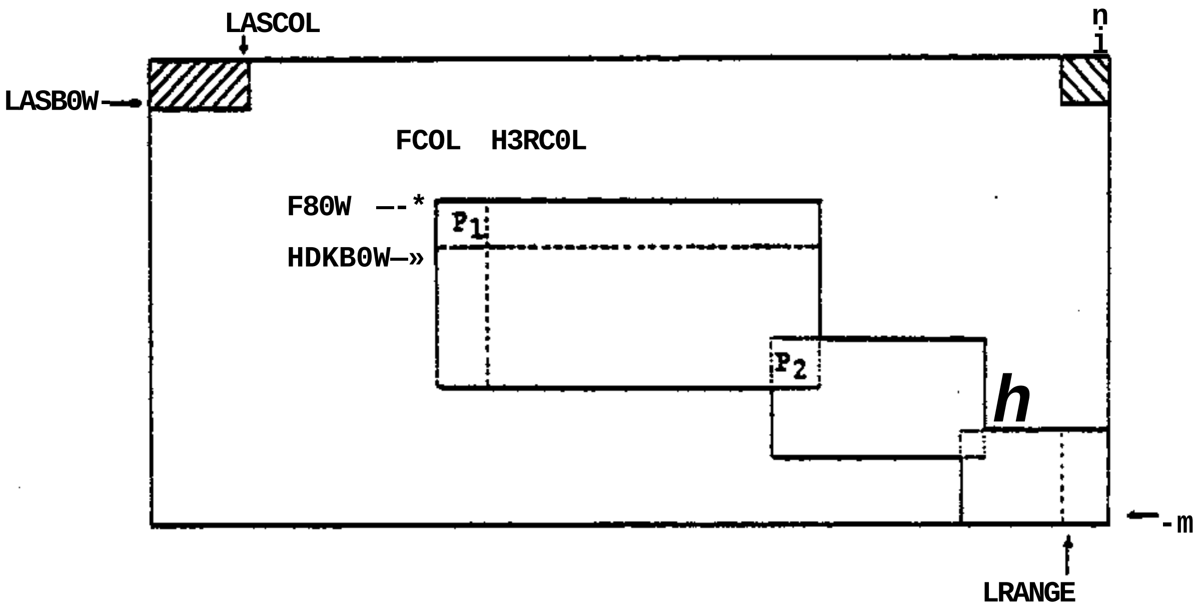


Figure 6: The resulting tableau window following horizon detection, stable data deletion, and augmentation. The indices F&OW and FCOL identify the first valid row and column; HORROW and HDRCOL, the lower bounds on row and column pivoting effects.

activity. If, as a result of pivoting, any column to the left of HORCOL or any row above HORROW are affected, then the missing data are affected by the pivot. Because the data are not available, we have a horizon violation and cannot solve the problem in core. By the same token, if no natural decomposition occurs, then we cannot detect a horizon, and the algorithm fails due to space limitations. Either of these conditions can occur as a result of a spike exceeding the allowable window size. When this happens, we have a block too large to fit into the finite tableau window. Such a problem must be solved with either a larger window, or other methods. In Appendix B we present a technique, disk swapping, to handle these cases when implementing the algorithm. The latter technique degrades the overall performance in some instances, but upgrades the forward simplex method so that it can solve more general problems.

To perform the augmentation, we increment T by 1, copy A_T into the upper left corner of the tableau, and B_{T-1} into the upper right. We now index the rows and columns in modular fashion. For the true indices of row i and column j we find their actual positions in the tableau by

$$(2)\quad \left\{ \begin{array}{l} i' = \mathrm{mod}(i-1,m) + 1 \\ j' = \mathrm{mod}(j-1,n) + 1 \end{array} \right.$$

We continue augmenting and pivoting until another horizon must be detected because of space limitations; i.e., an attempted augmentation would write over row FROW or column FCOL. Then we perform horizon detection, zeroing, and repeat the augmenting and pivoting. We continue in this manner until either $T = T_p$ or else a horizon violation occurs. Another valid stopping rule is to find a predetermined number of pinch blocks. This rule is not implemented in the first version of the code. We present a flow diagram of the[1] forward simplex method in Figure 7.

We hypothesize the forward simplex method should be able to solve typical stationary planning models (1) in linear time* This hypothesis is based on results for special structure models [4J, [12] and preliminary computational testing. If, for a problem, a natural decomposition due to a theoretical planning horizon is expected every q periods, then the expected amount of computations tshen T * q is doubled for T * 2q, tripled for T'«• 3q, and so on. Thus, even if the worst case bound for the q-period subproblem is encountered, we suspect that the problem decomposition into a series of q-period subproblems yields a linear time result. This is demonstrated in the computational results of the next section. In appendix A we discuss the computer implementation of the forward simplex method. In appendix B, we discuss refinements to the code.

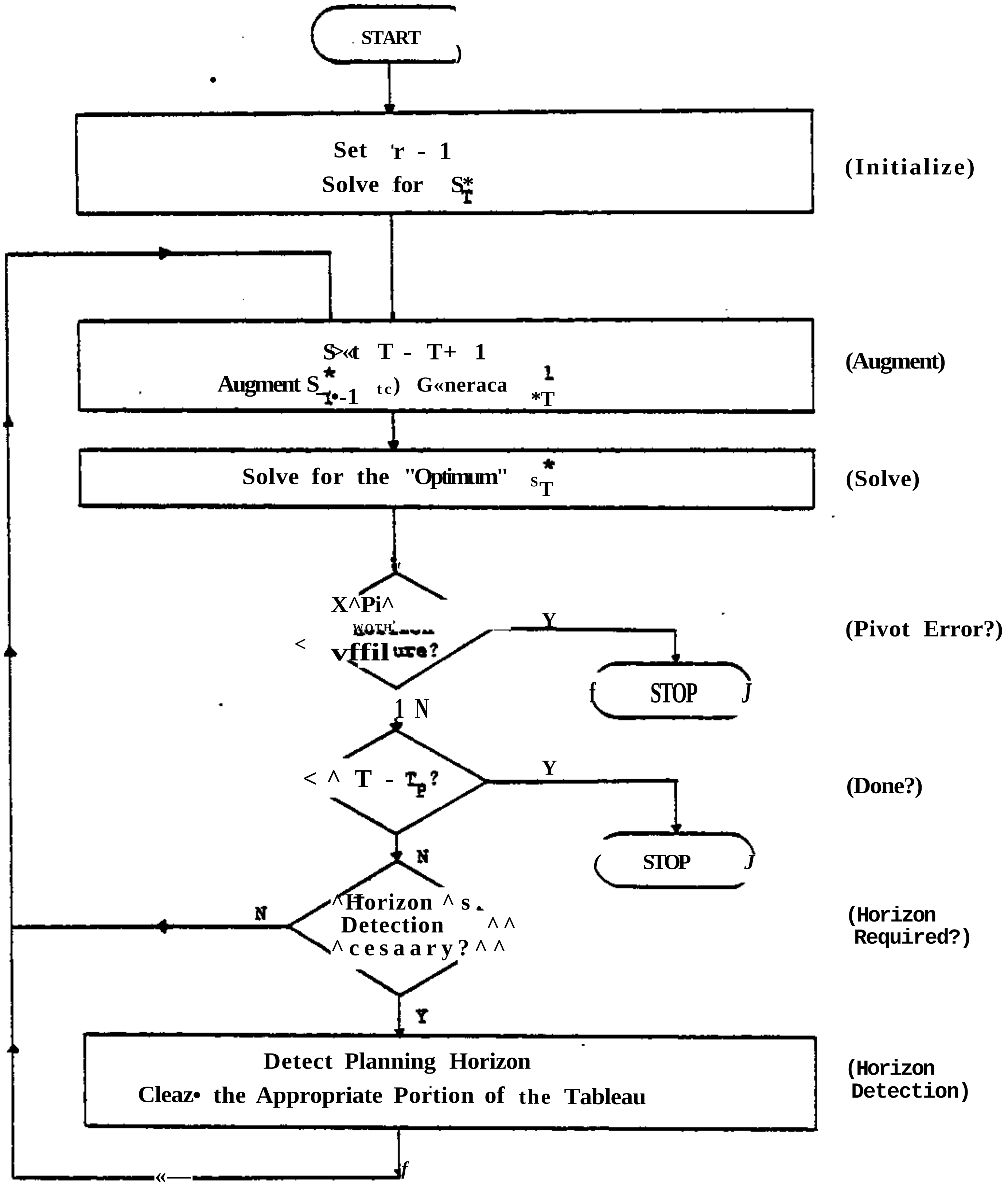


Figure 7: Flow Diagram of the Forward Simplex ttathod

5. Computational Results[2]

We tested the perf ormaace of the Forward Simplex Method on sets of randomly generated problems of three typical planning models. The results are extremely encouraging. Problems over ten times the size of the tableau window were solved. For all three models, both the solution time and number of pivots were found to be linear versus the problem length T. For example, even for very small problems that the standard LP code could solve, the standard code required 23 to 300 times longer solution times than the Forward Simplex Method* Detailed results follow.

The Forward Simplex Method was implemented in FORTRAN on the Carnegie-Mellon University DEC 20/60 B. The code can handle a maximum of 5000 time periods. The largest tableau window has dimensions 336 by 322. The max* imum dimensions of the A and B matrices are 22 by 22, so that a maximum of 14 periods of the largest A matrix fit. The condensed Tucker tableau [3] was used for ease of implementation. All input is read from a disk file. Output is printed onto a disk file, or the user's terminal. No basis reinversion is employed in the first version of the code. For comparison purposes, a standard linear programming code was developed from the forward code.

The three models tested were the production smoothing problem without inventory discussed in Aronson, Morton, and Thompson [4]; a similar model with the addition of inventory; and the manpower planning model discussed in Niehaus, Scholtz, *ao&* Thompson [15]. The characteristics of the models are suxnnarized in Table 1.

[2]This section is a brief susmary of Aronson [2].

In <u>all</u> cases, .for problems solved with the Forward Simplex Method, both the pivoting GPU time (a measure of solution time) and the number of pivots were found to be linear in the problem length 1. Even problems having dimensions exceeding ten times the tableau window size were solved in linear time. The 200 period manpower planning model required a mean of 43.43 seconds of CPU time to perform a mean of 3064.13 pivots to solve to optimality. These problems each had 2600 rows and 2600 columns. The conventional codes could only handle 312 rows and 312 columns. The mean pivoting CPU time and mean number of pivots for this set are shown in Figures 7 and 8. The wrap-around tableau window saves computer storage space. The maintenance of the staircase structure saves time. The performance of the Forward Simplex Method on this model is representative of the results found for the other two. All of these results are summarized in Table 2.

In Table 3, we summarize the results of the Forward Simplex Method applied to the largest problems for which the standard simplex code was tested•

The results for problems of each model solved with the Standard L? code are presented in Table 4. In all cases, the pivoting CPU time was found to be <u>at least cubic</u> in T. The number of pivots for the Standard LP code to solve the first model was $O(T^{1\cdot 507})$, while for models 2 and 3, it appeared linear.

We compare the results for the Forward Simplex Method to those of the standard code on the largest problems solved without wrap-around. The standard simplex method required about 28 to 500 times the CPU time required by the Forward Simplex Method to solve these small problems. For example, the Forward Simplex Method solved the 30 period model 1 problems in 1.25 seconds. The standard L? code required 124.93 seconds, about 100 times longer• It also required about 2 to 6 times the number of pivots. **These** results are sumarized in Table 5. Even for these small problems, the Forward Simplex Method proved to be far superior than the standard simplex method.

Model	Number of Problems	Demand Pattern	Matrix Size	Tableau Window *Size*	Maximum Number of * Periods in Window	Maximum Number of Periods Solved	Required Tableau Size	Number of Tableau Windows Used
1	21	Random	4 x 6	212x318	53	500	2000 x 3000	9.43
2	11	Cyclic Plus Random	5 x 7	230x322	46	500	2500x3500	10.87
3	11	Cyclic Plus Random	13x13	312x312	24	200	2600 x 2600	8.33

Table 1: Characteristics of the Three Models

Model	T* Maximum Number of Periods	Mean Pivoting CPU Time		Mean Number of Pivots	
		The Order of T	Seconds at T*	The Order of T	Number at T*
1	500	0(T)	**22.45**	0(T)	2820.19
2	500	otf)	**51.34**	0(T)	4768.00
3	200	0(T)	**43.43**	0(T)	3064.18

Table 2; Summary of Computational Results for the Forward Simplex Method. T* la the length of the longeat problems solved for each model. The mean pivoting CPU time ia the average of the problems in the aet. It la the time required to perform pivoting only. The order of T la the result of a regreaaion anaiyaia. The seconds at T* column indicates the mean pivoting CPU time for the longest problems of each model. A aimilar explanation holds for the mean number of pivot columns.

Model	Number of Periods **T***	Mean Pivoting CPU Time (Sec.)	Mean Number of Pivots
1	50	1.25	282.33
2	40	6.00	435.73
3	20	2.18	309.00

Table 3; Summary of the Forward Simplex Method's Performance on Problems of the same Size as solved with the Standard Simplex Method. These are used for comparing the two approaches.

Model	T* Maximum Number of Periods	Mean Pivoting CPU Time		Mean Number of pivots	
		The Order of T	Seconds at T*	The Order of T	Number at T*
1	50	$O(T^{3.5})$	124.93	$O(T^{1.507})$	614.85
2	40	$O(T^{3.097})$	169.54	$O(T)$	763.50
3	20	$O(T^{3.594})$	1092.03	$O(T)$	1941.83

Table 4: Summary of Computational Results for the Standard LP Code, The meaning of the columns Is Identical to that of Table 2.

Model	Number of Periods	Ratio of Standard Forward Code	
		Mean Pivoting CPU Time	Mean Number of Pivots
1	50	99.44	2.18
2	40	28.26	1.75
3	20	500.93	6.28

Table 5: Comparison for FORLP to STDLP. **The mean pivoting CPU time and number** of pivots at T* presented in Table **4** for **the standard LP code Is** divided by the same presented In **Table 2 for the forward code. This** Indicates that for the models tested, **the standard LP code required** about 28 to 500 times longer than **FORLP** to **solve these problems to** optlmallty. Also, It required **about** 2 **to 6 times more pivots.**

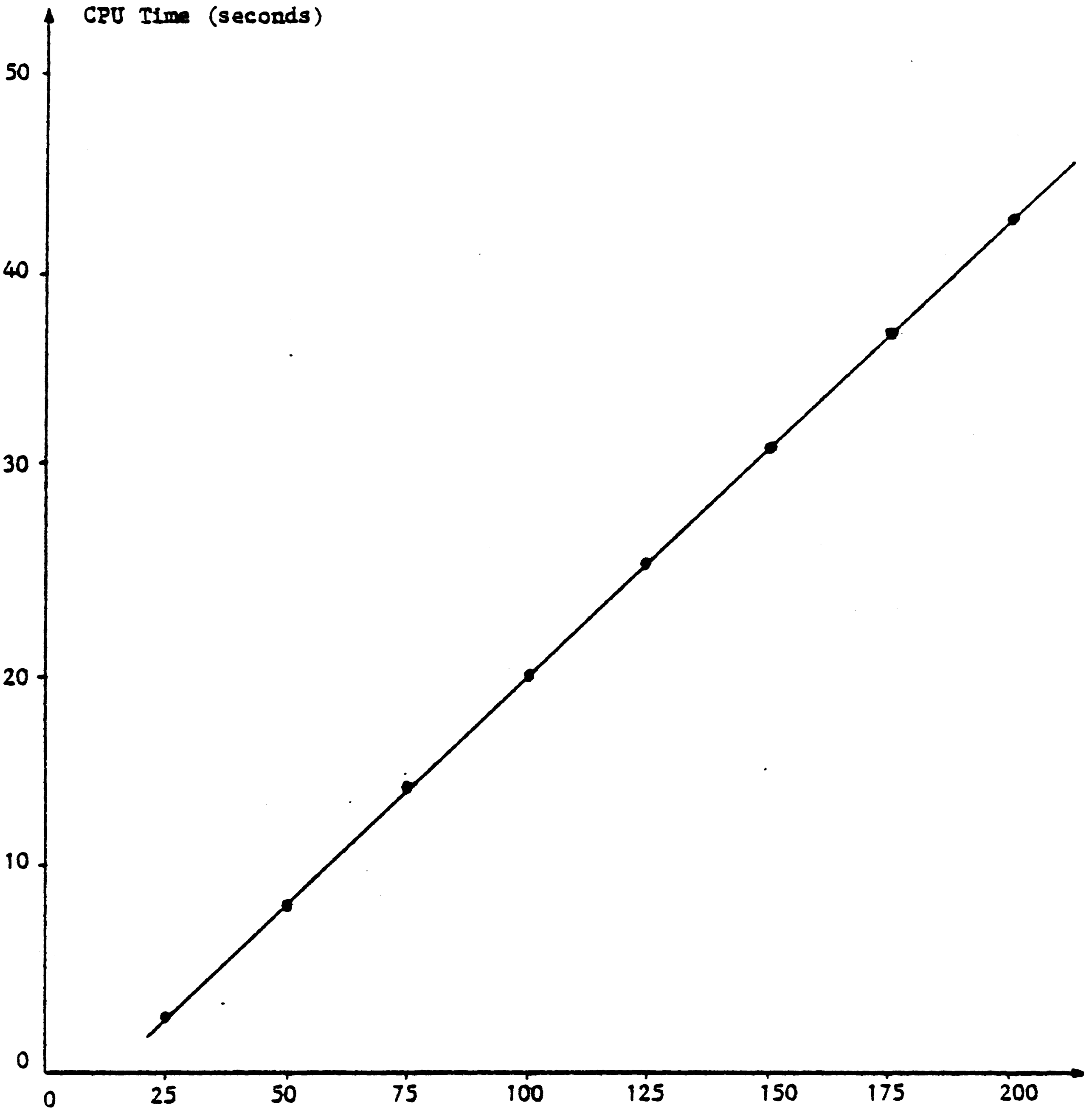


Figure 7: Mean Pivoting CPU Time versus T for 11 randomly generated manpower planning problems, solved with the Forward Simplex Method

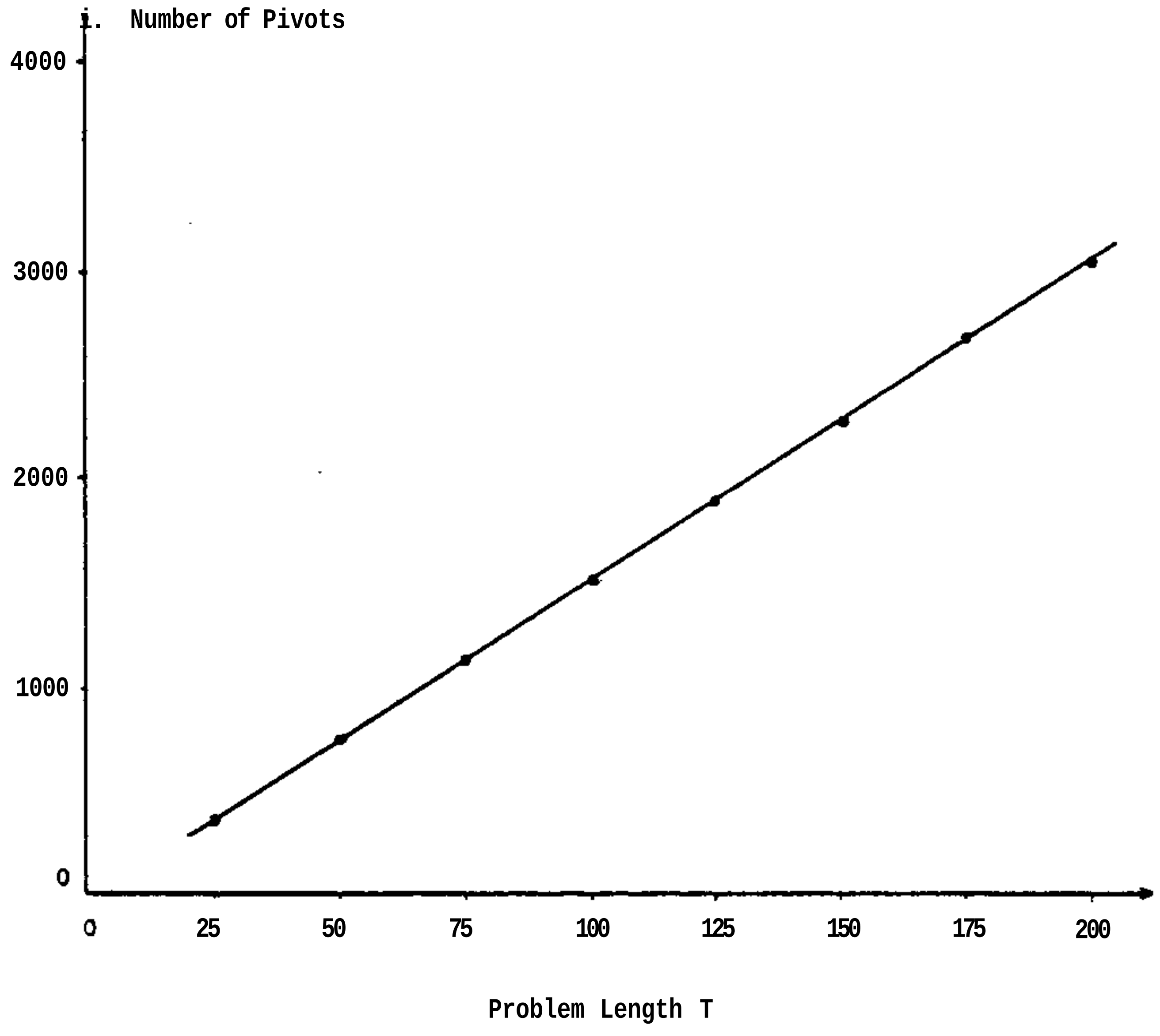


Figure 8: Mean Number of Pivoes versus T for ^1 randomly generated manpower planning problems, solved with the Forward Simplex Method

6. Conclusions

Tests suggest that the Forward Simplex Method solution time is linear is the problem length versus quadratic or cubic for earlier linear programming codes. Planning horizons due to the natural decomposition of staircase models are often found. In most planning situations, these horizons are useful in determining when a first period decision is good enough, thus saving computation. The extension of forward techniques to dynamic linear programming yields a powerful new method for solving large staircase structure problems efficiently-

Appendix A

Implementation

In this section we discuss the specific techniques implemented in the first version of the forward simplex method. We use the condensed Tucker tableau [3] for ease of implementation. This is a standard simplex tableau but without the identity matrix that corresponds to basic columns. At any time, the basis inverse can be generated if desired. This representation has been used successfully [8]. Although it is not efficient in terms of data storage, it is useful for testing purposes which involve ffxamining the tableau. Later versions of the code will utilize a compact form of the inverse.

We next develop the general methodology. A general linear programming problem, with a non-negative right hand side can be stated as

$$(3)\quad \begin{cases} \text{min} & ex \\ \text{subject to} & \\ & Ax \le d \\ & x \ge 0 \end{cases}$$

We simply negate inequality constraints of the form $A_ix \ge d_i$.

If a subset of the constraints of (3) are equality constraints:

$$(4)\qquad AjX - dj$$

We convert them to the form of (3) by the following transformation:

$$(5)\quad \begin{cases} \text{(a)} & A_1x \le d_2 \\ \text{(b)} & -eA_1x \le -ed_2 \end{cases}$$

Where e is a vector of ones. If v is the dual vector of (4), then $v = v^1 - v_1^1 \wedge e$, where v^1 is the dual vector of (5a), and ${}_1v^\wedge$ the dual of (5b).

We drop the nonnegativity restriction on b, choose K sufficiently large, and add the auxiliary variable x_{n+1} to (3) in the transformation to

$$
(6)\begin{cases}
 & \min \quad cx - K x_{n+1} \\
\text{subject to} & Ax - dx_{n+1} \le 0 \\
 & x_{n+1} \le 1 \\
 & x_i, x_{n+1} \ge 0
\end{cases}
$$

Clearly,

1. Problem (6) is feasible $(x, x_{n+1} = 0)$.
2. If (6) has an optimal solution with $x_{n+1} = 1$, then the same solution is optimal for (3).
3. If (6) has an optimal solution with $x_{n+1} < 1$, then (3) is infeasible.

We use the auxiliary x_{n+1} variable to simplify the attainment of a basic feasible solution. A slack starting basis is used. Any solution of (6) is not feasible to (3) until $x_{n+1} = 1$.

For the general dynamic linear program (1), we use an extra row for equality constraints (5b), and a local auxiliary variable x^t_{n+1} in <u>every</u> period. The tableau representation for periods t-1 and t appears in Figure A1. In examining test problems, we discovered that when a single equality row and auxiliary variable are used, the tableau becomes quite dense. Even for small models, data instability sometimes occurs. This also destroys the staircase structure and makes natural decompositions impossible to attain because all constraints in every period become interrelated. As a result, horizons cannot easily be detected in the tableau.

For implementation, we have three additional column vectors associated with the tableau. The first column is the true right hand side. The

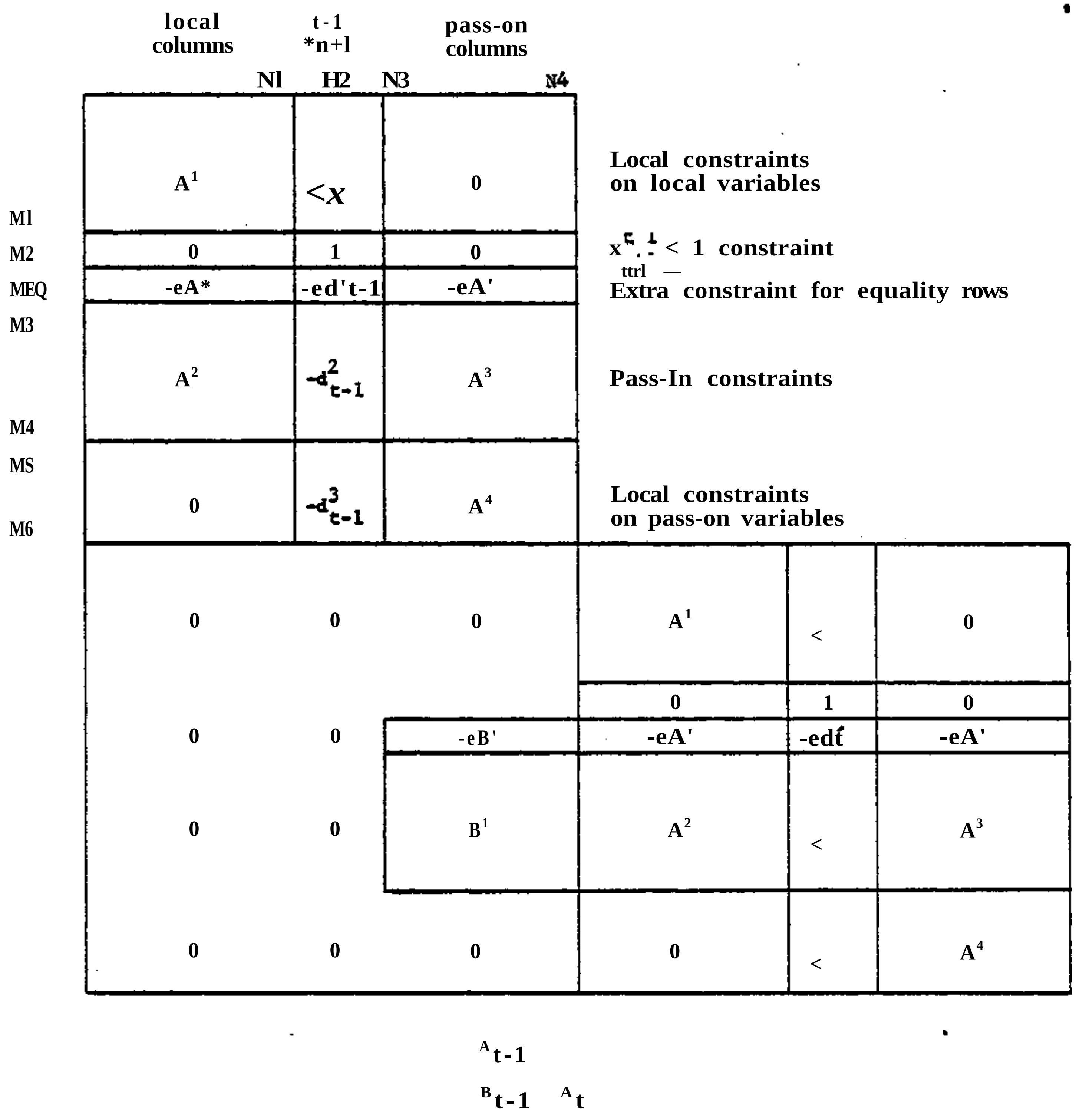


Figure Al: Condensed Tucker Tableau Representation of periods t-1 and t for the general staircase linear program, A primed vector or matrix means that only the rows corresponding to the equality constraints are included. The vector d_t is also partitioned into 3 parts.

second is a perturbed right hand side to break ties and prevent cycling. (We could also have used Bland's rule [51, or other anti-cycling devices [7], [21].) The third column is an extra one used for pivoting. As the pivot column is transformed to a unit vector, this third column becomes the new column associated with the variable leaving the basis. A separate row vector is maintained for reduced costs, and a separate element for the total cost, We use a separate vector to improve the reduced cost scan. For an array element, three memory accesses are required; for a vector, only two. We store the row and column indices in vectors, using the scheme that variable i in period t has index 100t+-i.

Data entry is from a disk file. The A and B matrices are read and converted to the form shown in Figure Al. The first period problem is initialized with a slack basis, and pivoting is performed until S_i^* is found. Care must be taken to specify A and B so that each T period sub* problem is feasible with the pass-on variables set to zero. If $x^1_{n+i} < 1$, computation stops because the subproblem is infeaaible. Otherwise, augmentation and pivoting continue as outlined in section 4.

When the entire tableau window is full we detect horizons, discard stable data, and employ wrap-around. At this point FROW » FCOL * 1, LASB0W * m, and IASCOL * n« The adapted A matrix has dimensions M6 X N4. Let ISMN_i be the first non zero entry of row i, ISMAX_i the last. Similarly, define JSMIN_j and $\text{JSH\&X}_j$ for column j.

We define

$$(7)\quad \left\{ \begin{array}{ll} \text{IHMIN}_i * \min\limits_{\text{fc»i,...,LASSOW}} \text{flSMD}k_J; & \text{- i-FBOW,..., LASROW} \\ \\ \text{JHMIN}_2 - \min\limits_{\text{k-j,...,LASCOL}} (\text{JSMIN}_k\}; & \text{j*FCOL,..., LASCOL} \end{array} \right.$$

To datect a horizon, we first recursively find IHKIN and JHMIN by the formulas:

$$(8) \quad \begin{cases} \text{IHMIN}_i = \text{Min } [\text{iHMn}_i\text{r\^{}ISMIN\^{}} ; \ \text{i\^{}FSOW}, \ldots, \text{LASROW-1} \\ \\ \text{JHMIN}_j * \text{Min } \{\text{JHMIN\^{}JSMIN}_j\}; \ \text{j\^{}FOOL}, \ldots, \text{LASCOL-1} \end{cases}$$

Then, we scan the rows from FBOW to LASROW until a $\text{THMTN}_i \geq \text{N4}$, and the columns for $\text{JHMIN}_j \geq \text{MS}$. In essence, we find a series of blocks large enough to augment at least one period.

If we detect no horizon, then the tableau window is too small and the problem overflows. In the next appendix we discuss techniques to handle this case. Otherwise we print the partial solution of the stable block onto two disk files, one for the primal, the other for dual information. When the entire problem is solved we reconstruct the solution from these files. We clear out the block in the tableau. We set BORROW to the row i for which IHMXN_i is found to be a pinch block; BDRCOL to the column j for which JHMIN_j is found. We reset FBOW to *JEtG^^CQl* *"* FCOL tO IHM:Dl aOBB0W[1] and augment as shown in Figure 6. We now index the tableau with the mod functions (2). As described in section 4, if while pivoting JSMEL_u < BORROW or ISTCEL_k < BDRCDL then a horizon violation occurs. Later versions of the code will handle the overflowing of die tableau window. The current version simply stops.

In determining subsequent horizons, we consider the true difference in position between IASCOL and IHMIN_i to exceed N4, and between LASROW and JHMIN_j to exceed M6. This guarantees enough space in the tableau window for at least one augmentation.

For all problems we assume that the data are appropriately scaled, all lower bounds on pass-on variables are 0, and that the constraint matrices are partitioned as described in section 2.

We employed <u>no</u> basis reinversion for this first version. The natural decomposition of (1) effectively blocks numerical errors from rippling from early periods into later ones, so that we did not encounter numerical difficulties on the test problems tried. The forward simplex method performs automatic spike reduction, thus we do not reorder the basis to maintain a diagonal tableau ([9], [10]). The staircase structure is an inherent feature of the multi-stage model. The forward simplex method merely maintains as much as possible of the structure.

The first version of the forward simplex method was written in FORTRAN and developed on the Carnegie-Mellon University DEC 20/60 B. It requires 256K of addressable core. The global data require 224K, leaving 32K for local data. The code can handle up to 5000 time periods. The maximum dimensions of the A and B matrices are 22 by 22. The tableau window is dimensioned 336 by 322, so that 14 periods of the largest A matrix fit. In the interest of programming in "standard" FORTRAN, all do loops increment. In appendix B we discuss improvements for future versions of the code. A detailed computational study of the code is in the thesis by Aronson [2].

Appendix B

Improvements

There are four immediate improvements that we plan to implement in later versions of the forward simplex method. These include data packing, updating the code so that horizon failures do not occur, not outputing intermediate solutions to disk files, and not checking for infeasibility at each T period subproblem*

The first improvement is the data packing. The first version uses a condensed, but complete tableau window. Because of its sparseness, a packing scheme of some kind should be implemented in later versions. The simplest kind is row packing. We represent every row i of the tableau as $ISHIN_i$, $ISWkX_t$, the number of entries $ISMAX_L$ - $ISMHT_1$ + 1, followed by the entries. More advanced techniques, such as an L-U decomposition of the basis, might also be applied.

The issue of horizon failure due to tableau window overflow affects problem solvability. Instead of discarding the stable portion of the tableau after horizon detection we could store it in a disk file. Then, whenever a horizon violation occurs while pivoting, we could retrieve the presumed stable portion of the tableau. We can adapt this idea for use when horizons cannot immediately be f012nd. In this case, all pivoting is performed in a random access disk file, until a new planning horizon occurs. This technique is basically the same as virtual core/disk swaps, performed by computer software. This swapping techñique will slow down the overall performance when horizon violations occur, or horizon existence cannot be determined. On the other hand, it will improve the code so that it can solve all general dynamic linear programs. An interesting feature to add to this is automatic tableau window expansion. That is, if a larger tableau window is required, then let the code change the window size to accommodate the problem. Since this feature is machine dependent, we shall investigate the possibility of implementing it at a later time.

The third improvement involves not writing the intermediate solution onto disk for subsequent retrieval, reconstruction and printing. Instead of storing the whole solution, we could simply reconstruct as much as is possible and print it directly. We could store on disk that portion of the partial solution that needs data from a later block to complete the reconstruction. This improvement reduces the size of the disk files and eliminates most of the disk I/O which is performed by the first version of the code. This would also allow us to solve problems longer than 5000 periods.

The last improvement makes the code more general. The current version requires that each T period subproblem be feasible with the pass-on variables non basic at zero. If we only check the values of $x^t_.$, in the stable blocks, then we would allow primal infeasibility to occur except for stable data, and in the last period. This improvement allows for positive lower bounds on the pass-on variables.

We shall consider all of the improvements discussed in the development of new versions of the code.

References

[I] Aronson, J.E., "Investigation of Further Properties of die Production Smoothing Problem without Inventory", W.P. #61*79-80, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, April 1980,

[2] Aronson, J.E,, "The Forward Simplex Method: Computational Results", W.P. #62-79-80, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA*, April 1980.

[3] Aronson, JJS., Forward Linear Programming, Ph.D. Thesis, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA., April 1980.

[4] Aronson, J.E., Morton, T.E., and Thompson, G.L., "A Forward Algorithm and Planning Horizon Procedure for the Production Smoothing Problem without Inventory", W.P. #20-78-79, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA., Uovember 1978.

[5] Bland, R.G., "Hew Finite Pivoting Rules for the Simplex Method", Mathematics of Operations Research, 2, 1977, 103-107.

[6] Charnes, A., "Optimality and Degeneracy in Linear Programming", Ecoooraetrika, 20, 2, 1952.

[7] Dantzig, CB., Linear Programming and Extensions, Princeton University Press, Princeton, NJ., 1963.

[8] Gaver, D*P», and Thompson, G.L., Programing and Probability Models in Operations Research, Brooks Cole Publishing Co., Monterey, CA., 1973.

[9] HeHerman, E., and Rarick, R., "The Partitioned Preassigned Pivot Procedure (P^)", in Sparse Matrices and their Applications, Edited by Rose, D. and Willoughby, R., Plenum Press, New York, NY., 1972, 67-76.

[10] HeHerman, E., and Rarick, R., "Reinversion with the Preassigned Pivot Procedure", Mathematical Programming, 1, 1971, 195-216.

[II] Kunreuther, H., and Morton, T.E., "Planning Horizons for Production Smoothing with Deterministic Demands: I, II", Management Science, 20, 1973, 1974, 110-125, 1037-1046.

[12] Lundin, R.A*, and Morton, !•£•, "Plaxming Horizons for the Dynamic Lot Size Model: Zabel vs. Protective Procedures and Computational Results", Operations Research, 23, 4, July-August 1975, 711-734.

[131 Miller, L,W., "Using Linear Programs to Derive Planning Horizons for a Production Smoothing Problem", Working Paper 79-10-03, Department of Decision Sciences, The Wharton School, University of Pennsylvania, July 17, 1979 (to appear in Management Science),

[14] Morton, I.E., "Forward Algorithms for Forward Thinking Managers", W«P. #7-78-79, Graduate School of Industrial Administration, Carnegie-Mfellou University, Pittsburgh, PA., August 1978,

[15] ftiehaus, R.J«, Scholtz, D., and Thompson, 6,L., "Managerial Tests of Conversational Manpower Planning Models," TIMS Studies in the Management Sciences, North-Holland Publishing Co., 8, 1978, 153-171.

[16] Perold, A.F., and Dantzig, G.B., "A Basis Factorization Method for Block Triangular Linear Programs", Technical Report SOL 78-7, Systems Optimization Laboratory, Department of Operations Re-Search, Stanford University, Stanford, CA., April 1978. •

[17] Sethi, S. and Chand, S., "Planning Horizon Procedures for Machine Replacement Models", Management Science, 25, 2, February 1979, 140-151.

[18] Simmonard, M., Linear Programming, Prentice Hall, NJ., 1966.

[19] Thompson, G.L., and Sethi, S.P., "Turnpike Horizons for Production Planning", W»P. #29-77-78, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA., March 1978.

[20] Wagner, H.M- and Whitin, T.M., "Dynamic Version of the Economic Lot Size Model," Management Science, 5, 1958, 39-96.

[21] Wolfe, P., "A Technique for Resolving Degeneracy in Linear Programming", RAM) Report RM-2995-PR, The RAND Corp., Santa Monica, CA., 1962.