APPLICATION OF STATE INCREMENT DYNAMIC PROGRAMMING TO INDUSTRIAL MANAGEMENT SYSTEMS

by 45

RAMA NAND KAPUR

B.Tech.(Honors), Indian Institute of Technology

Kharagpur, India, 1957

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Industrial Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1969

Approved by:



Major Professor

LD
2668
R4
1969
K36

TABLE OF CONTENTS

1. INTRODUCTION..... 1

2. CONVENTIONAL DYNAMIC PROGRAMMING..... 4

Multistage Processes..... 4

Continuous Processes..... 6

Computational Procedure..... 7

3. DIMENSIONALITY DIFFICULTIES..... 11

Polynomial Approximation..... 11

Lagrangian Multipliers..... 13

Ratio of State Variables..... 13

Quasilinearization..... 14

State Increment Dynamic Programming..... 15

4. STATE INCREMENT DYNAMIC PROGRAMMING..... 16

Formulation of the Problem..... 17

Computational Procedure..... 18

Comparison with Conventional Dynamic Programming..... 25

5. APPLICATIONS..... 29

A Production Scheduling Model with Learning Effect..... 29

An Inventory Model with One State Variable.... 35

a. Solution by Conventional Dynamic Programming..... 37

b. Solution by State Increment Dynamic Programming..... 38

c. Comparison..... 52

An Inventory and Advertising Model with Two State Variables..... 56

a. Solution by Conventional Dynamic Programming..... 60

b. Solution by State Increment Dynamic Programming..... 68

c. Comparison..... 70

Discussion..... 78

6. ACKNOWLEDGEMENT..... 81

7. REFERENCES..... 82

Appendix A: Flowcharts..... 84

Appendix B: Computer Programs and Results..... 87

SYMBOLS

f	scalar function
$\underline{f}$	vector function
h	profit function
s	dummy variable for time
t	time, independent variable
t_o, t_f	initial and final time respectively
x	scalar state
$\underline{x}$	state vector
z	scalar control
$\underline{z}$	control vector
B	block designation
$G^{(k)}$	profit resulting from control $z^{(k)}$ over incremental time
$H^{(k)}$	interpolated optimal profit at state $x^{(k)}$
$I(x,t)$	optimal profit at state x and time t
$J(x,z,t)$	profit function
K	total number of control increments
M	total number of time increments
S	total number of time increments Δt within a block
Z	set of admissible controls
X	set of possible states
α^-, α^+	bounds on control variables

β^-, β^+	bounds on state variables
Δx	fixed increment in state variable
Δt	fixed increment in time
ΔT	interval in independent variable time covered by a block
Δz	fixed increment in control variable

1. INTRODUCTION

Dynamic programming is a useful tool in optimizing multistage optimization problems. The method of dynamic programming is based on the mathematical notion of recursion. The main characteristics of optimization problems to which it can be applied is that several decisions have to be made to optimize the overall performance of a system. Furthermore, the system consists of several stages and the decision made at the later stage do not affect the performance of the earlier stages.

For a multistage process, the optimal overall policy cannot be arrived at by considering the effect of each decision at each stage separately. In other words, it is not true that the overall optimum is the same optimum obtained by optimizing each individual stage. It might, for example be that a bit of sacrifice in the first stage may place us in a much stronger position in the second stage and thus a higher total profit is obtained. This will be achieved if the method of dynamic programming is used.

The subject owes much of its development to Bellman and his associates at the Rand Corporation. The concept on which dynamic programming [1-5] is based is the principle of optimality which states:

> "An optimal policy has the property that, whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with respect to the state resulting from the first decision".

This concept is a new and versatile mathematical tool for the treatment of many practical and novel problems. Because of the simplicity and the versatility of the principle of optimality, the method of dynamic programming can be used to handle those cases for which the tranformation at each stage is too complex to be expressed analytically or involves non-differentable functions. A continuous process can also be treated by breaking it into a large number of infintesimal stages.

Dynamic programming works best when the number of state variables is not too large. The resaon is that the number of fast memory storage locations required for computation increases exponentially with the increase in the number of state variables. In general $2N^n$ such locations are required, where n is the number of state variables and N equals the number of intervals into which each state variable is divided. The average number of fast memory locations available on a modern computer is 50,000. Thus if there are three state variables, 200,000 locations would be required which far exceed the limit. This extremly large fast memory requirement has been called by Bellman as the "curse of dimensionality". This requirement limits the use of dynamic programming to relatively simple cases. Various schemes have been suggested to overcome this dimensionality difficulty. However, most of these schemes trade computer time for computer memory. As such they are not fully satisfactory.

Recently a computational scheme called state increment dynamic programming [8,9] has been developed which considerably reduces memory requirements without increasing the computation time. The purpose of this report is to apply this procedure to industrial management systems in the production planning area. In the problems considered, both the conventional and the new methods are used. The results obtained in both cases are compared. The advantages as well as the disadvantages are highlighted.

2. CONVENTIONAL DYNAMIC PROGRAMMING

As mentioned earlier, the corner-stone of dynamic programming is the principle of optimality[2-4]. The mathematical transliteration of this principle yield the necessary functional equations. Any process to be considered can be either a discrete multistage process or a continuous process. The derivations of the functional equations are briefly reviewed in the following.

MULTISTAGE PROCESSES

A typical multistage process is shown in Fig. 1. Vector $\underline{x}$ represents a set of state variables. Vector $\underline{z}$ represents a set of control variables. The stages are numbered in a backward fashion for convenience. By applying control $\underline{z}_n$ at stage n, the incoming state $\underline{x}_{n+1}$ is transformed into the outgoing state $\underline{x}_n$. This tranformation is expressed as

$$\underline{x}_n = T_n (\underline{x}_{n+1}, \underline{z}_n), \qquad n=N,N-1,.....2,1 \qquad (1)$$

Associated with each stage there is a return and this return is expressed as $g_n(\underline{x}_{n+1}, \underline{z}_n)$. The sum of all the stage returns is called the return function S. Our objective is to find a set of permissible values $\underline{z}_1$, $\underline{z}_2$,....., $\underline{z}_N$ so that the return function is maximized (or minimized).

$$S(\underline{x}_{N+1}, \underline{z}_N, ..., \underline{z}_2, \underline{z}_1) = \sum_{n=N}^{1} g_n(\underline{x}_{n+1}, \underline{z}_n) \qquad (2)$$

Consider the stage n as the initial stage at which

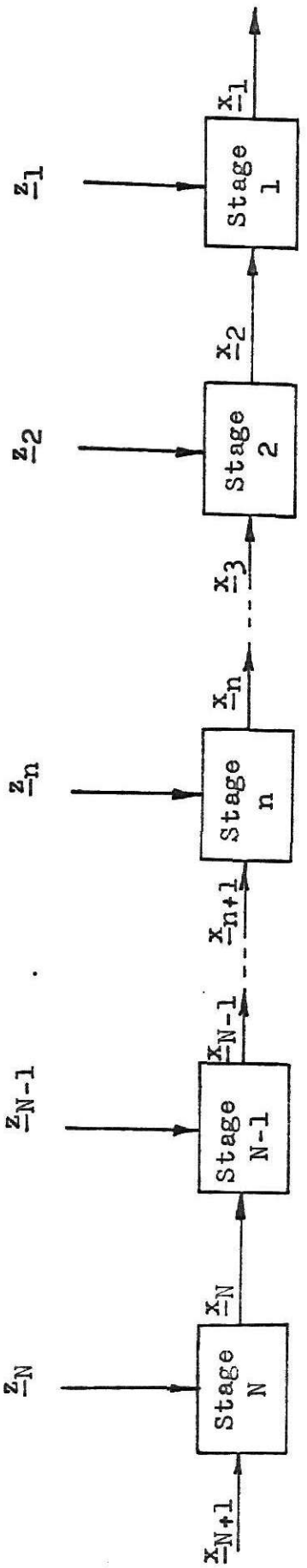


Fig. 1. A typical multi-stage process.

a control is to be applied. Define a function

$f_n(\underline{x}_{n+1})$ = the value of the objective function for the last n stages when the optimal policy is followed and when the incoming state at stage n is $\underline{x}_{n+1}$.

Then it follows from the principle of optimality that

$$f_n(\underline{x}_{n+1}) = \max_{\underline{z}_n} \left\{ g_n(\underline{x}_{n+1}, \underline{z}_n) + f_{n-1}(\underline{x}_n) \right\} \tag{3}$$

This is the functional equation giving a recursive relationship between an n stage process and an (n-1) stage process. A complete derivation of this equation can be found in any text book on dynamic programming [1-4,7,14].

CONTINUOUS PROCESSES

In the continuous case, the system is assumed to vary continuously with respect to an independent variable t. In the production planning problems to be discussed later, the independent variable is time. Changes in the system occur according to a set of differential equations given by

$$d\underline{x}/dt = \underline{f}(\underline{x}, \underline{z}, t) \tag{4}$$

Eqn. (4) can be reduced to the difference equation

$$\underline{x}(t+\Delta t) = \underline{x}(t) + \underline{f}(\underline{x}, \underline{z}, t)\Delta t \tag{5}$$

The return function J is given by the integral

$$J = \int_{t_o}^{t_f} h(\underline{x}, \underline{z}, s)ds \tag{6}$$

where h = profit (or cost) function per unit time

s = dummy variable for integration

t_o, t_f = initial and final time respectively

Our objective is to find the policy $\underline{z}(t)$ which maximizes J, with the initial condition being given as $\underline{x}(0) = \underline{c}$. For simplicity, let us consider a one-dimensional case. Clearly when the optimum function J has been obtained, the integral is a function only of the initial condition c and the duration of the process t_f. Define

$g(c,t_f)$ = the maximum value of J where the starting state of the process is c and the total duration is t_f.

By using the principle of optimality it can be shown that

$$g(c,t_f) = \max_{z(0,\Delta)} \left[h(c,z(0))\Delta + g(c + f(c,z(0))\Delta , t_f - \Delta \right) \qquad (7)$$

where, Δ is incremental time t. This is the desired functional relationship. Similar equations can be obtained for the multi-dimensional case [2-4,11].

COMPUTATIONAL PROCEDURE

To illustrate the procedure, the problem represented by equations (4) to (7) will be considered. For simplicity, the vectors $\underline{x}$ and $\underline{z}$ will be assumed one-dimensional. In practical problems, the range of state and control variables is either known or can be estimated. For instance, if the

state variable x represents the inventory of a warehouse, then x must range between zero and some finite value. Thus we write

$$\beta^- \leq x \leq \beta^+$$

$$\alpha^- \leq z \leq \alpha^+$$

The state variable x is quantized with an increment of Δx, the quantized values being expressed as

$$x = \beta^- + j\Delta x, \quad j = 0,1,.....N \tag{8}$$

where $N\Delta x = \beta^+ - \beta^-$

The set of all possible values of x is denoted by X. Similarly the set of all possible values of z is denoted by Z. Its quantized values are given by

$$z = \alpha^- + j\Delta z, \quad j = 0,1,.....K \tag{9}$$

where $K\Delta z = \alpha^+ - \alpha^-$

The set Z can be represented as

$$Z = \left\{ z^{(1)}, z^{(2)},.....z^{(k)},.....,z^{(K)} \right\}$$

For a continuous process, the independent variable t is quantized as

$$t = t_o + m\Delta t, \quad m = 0,1,.....M \tag{10}$$

where $M\Delta t = t_f - t_o$

To initiate the computation, the value of optimal profit at $t = t_f$ for all quantized state $x \in X$ is determined and stored as a table on the fast memory of the computer. Next

a quantized state $x \in X$ at time $t = t_f - \Delta t$ is taken. By applying each of the admissible controls $z^{(k)}$, the resulting state $x^{(k)}$ and the profit $G^{(k)}$ over the interval Δt are evaluated and can be written as

$$x^{(k)} = x + f(x, z^{(k)}, t_f - \Delta t)\,\Delta t \tag{11}$$

$$G^{(k)} = h(x, z^{(k)}, t_f - \Delta t)\,\Delta t \tag{12}$$

The optimal profit at $t = t_f$ for each of the states $x^{(k)}$ is found as follows. The table of values of optimal profit for all $x \in X$ at time $t = t_f$ is stored in the fast memory. However, in general the state $x^{(k)}$ will not lie on one of the quantized values at which the optimal profit is stored; rather it will fall between two such values. Linear interpolation based on these two values usually gives sufficient accuracy.

Let $I(x,t)$ denote the optimal profit at state x and time t and let $H^{(k)}$ denote the interpolated value of the optimal profit at state $x^{(k)}$. Then

$$H^{(k)} = I\,(\,x^{(k)},\, t_f\,) \tag{13}$$

Adding eqns. (12) and (13) gives

$$G^{(k)} + H^{(k)} = h(x, z^{(k)}, t_f - \Delta t)\Delta t + I(x^{(k)}, t_f) \tag{14}$$

But this is exactly the quantity which is to be maximized by the choice of $z^{(k)}$. This maximization can be achieved by the use of a suitable search technique.

The above procedure is repeated for all values of x at

$t=t_f-\Delta t$. The value of optimal profit $I(x, t_f-\Delta t)$ are thus obtained and stored in the fast memory. The values of the optimal profit at $t=t_f$ are no longer required and are hence printed out.

Next, the entire procedure is repeated for $t = t_f-2\Delta t$ for all the quantized values of x. In this case the optimal profits at $t = t_f - \Delta t$ are used for interpolation. These steps are then carried out untill the time $t = t_o$ is reached, at the end of which the optimal controls and optimal profits will have been computed for all values of x and t. This complete set of data gives the solution of the problem for all possible initial values of x.

A simplified flowchart giving the steps involved in the computational procedure appears in appendix A, Fig. 19.

3. DIMENSIONALITY DIFFICULTIES

The number of fast memory locations needed in the conventional dynamic programming is proportional to the total number of discrete states. If there are n state variables and N_i quantization levels of the ith state variable, then the total number N of all possible states is given by

$$N = \prod_{i=1}^{n} N_i$$

With three state variables and $N_i = 100$, N equals 100^3 or 10^6. However, the modern computer has a fast memory capacity of the order of 10^4 to 10^5. Therefore, a three-dimensional problem cannot be solved routinely. Bellman has coined this difficulty as "the curse of dimensionality" [2-4].

Various techniques have been proposed to overcome this difficulty. Essentially such techniques trade computer time for computer memory. One simple technique is to simply increase the increment size. This, however, often results in discretization error; hence, it is seldom used. In the rest of this section, various other techniques are briefly discussed.

POLYNOMIAL APPROXIMATION

In this technique the table of optimal return function is correlated into a polynomial during computation. One

way to do this is to approximate the optimal return function $f_n(x)$ by a truncated power series of R+1 coefficients. Thus

$$f_n(x) = \sum_{k=0}^{R} a_k . x^k$$

However, instead of a truncated power series, the use of orthogonal polynomials such as, Legendre polynomial is recommended. The later has several advantages over the former. First, a truncated power series is accurate only around the point of expansion and the accuracy decreased rapidly on moving away from this point. Legendre polynomials, however, give uniform accuracy. Second, the coefficients of Legendre polynomials are found by numerical integration, whereas, numerical differentiation is used for the power series. The later process is inaccurate. Third, a simple reccurence relationship can be used for determining the Legendre polynomials.

Using this technique for a one-dimensional problem, one needs to store only R+1 coefficients in the fast memory instead of say, 100, storages needed in the conventional approach. This difference is further enhanced in problems of larger dimension.

The major drawback of this technique is, however, that it has proven to be of value for a class of "well-behaved" problems where the optimal return function can be correlated accurately by a low order polynomial. For more complicated problems, use of a sufficiently high order polynomial to

get the required accuracy results in excessive computation time.

LAGRANGIAN MULTIPLIERS

Lagrangian multipliers are used extensively in optimization problems with inequality constraints. By introducing a number of Lagrangian multipliers, it has been shown [4-5,13] that a reduction in the number of state variables can be achieved. However, as a result of this exchange, one must search for these added multipliers. This is essentially trading computer memory for computer time. Furthermore, when a problem involves non-differentiable functions, difficulties may arise in using Lagrangian multipliers. This is particularly important since many practical problems contain functions of discrete variables which are not differentiable. Another disadvantage is that with the introduction of these multipliers, global optimum may not be obtained.

RATIO OF STATE VARIABLES

In some processes the tranformation functions of the state variables are of homogeneous nature and are linear with respect to the state variables. Such properties simplify the problem in that the ratio of two state variables may be used as a new state variable. In this way a reduction in the number of state variables is achieved.

QUASILINEARIZATION

Consider the problem of optimizing the function

$$h(x_1(t_f),\ x_2(t_f),.....,x_m(t_f) \qquad (1)$$

over the control variable $\underline{z}(t)$, which are related to the state variables $\underline{x}$ by non-linear differential equations

$$dx_i/dt = f_i(\underline{x},\ \underline{z}), \qquad i = 1,2,.....M \qquad (2)$$

and with the initial conditions

$$x_i(0) = x_i^o, \qquad i = 1,2,.....M \qquad (3)$$

where, $m < M$, $t_o \leq t \leq t_f$, and $\underline{x}$ and $\underline{z}$ are M-dimensional vectors. In addition, the following constraints must be satisfied.

$$\alpha_i^- \leq z_i(t) \leq \alpha_i^+, \qquad i = 1,2,.....M \qquad (4)$$

$$\int_{t_o}^{t_f} f(\ \underline{z}\ )\ dt \leq L \qquad (5)$$

If the functional equations of conventional dynamic programming are used, M variables are involved. However, Bellman and Kalaba [3] have shown that if eqn. (2) were linear, the above problem could be treated by dynamic programming involving sequences of m variables. Since m is equal to one or two for a number of significant problems, there is a very important reduction in the dimensionality of the problem. This is especially true in view of the fact that eqn. (2) can be linearized by the quasilinearization

procedure.

For a comprehensive and yet simple treatment of the above-mentioned four techniques, the reader is referred to Lee [10,11,13].

STATE INCREMENT DYNAMIC PROGRAMMING

This technique was recently developed by Larson [8,9]. It differs from the conventional dynamic programming in the choice of the time interval over which control is applied. Instead of using a fixed time interval, this technique uses the minimum time interval required for at least one of the state variables to change by one increment. The next state, therefore, lies within a small neighborhood of the point at which control was applied. This fact is exploited to reduce the fast memory locations requirement.

Since the main purpose of this report is to apply this technique to industrial management systems, the next section treats it in greater details.

4. STATE INCREMENT DYNAMIC PROGRAMMING

The state increment dynamic programming technique provides a good method for reducing the fast memory requirements. The reduction in the fast memory requirement is enormous, usually several orders of magnitude. Furthermore, this method retains all the desirable properties of the conventional dynamic programming. For example, constraints of a wide variety including inequality constraints are handled with the same ease as in conventional dynamic programming. The property that a solution is obtained for all possible initial conditions is also retained. The total computing time is of the same order of magnitude when compared with conventional dynamic programming. One limitation of this method is that it can only be applied to continuous processes.

The basic difference between this and the conventional method lies in the choice of time interval over which control is applied. In the conventional method, this interval is fixed. In the new method, however, this interval is the minimum time required for at least one of the state variables to change by one increment. This is the motivation for calling this method state increment dynamic programming.

As a result of the above choice of time interval, the next state will lie in the neighborhood of the given state. Specifically,with n state variables, the next state will lie on the surface of an n-dimensional hypercube centered at the

given point and with length of $2\Delta x_i$ along the ith co-ordinate, where Δx_i is the incremental size of the ith state variable. This fact is exploited in reducing the fast memory requirements as follows. The entire state-time space X-t is partitioned into several blocks. Each block covers only few increments along each state but relatively long interval of time. Optimal control is then computed for one block at a time and not for the entire state space X as is done in the conventional method. For a problem involving three state variables each with 100 quantization levels, the conventional method requires 100^3 or 10^6 fast memory locations. In the new method, if each block contains three quantized values for each state variable, only 111 locations are needed. Clearly this is a very significant reduction.

FORMULATION OF THE PROBLEM

The same problem treated in section 2 will be considered. The system equations are non-linear differential equations given by

$$\frac{d\underline{x}}{dt} = \underline{f}\,(\underline{x},\ \underline{z},\ t) \tag{1}$$

where, $\underline{x}$ = n-dimensional state vector

$\underline{z}$ = q-dimensional control vector

Rewriting eqn. (1) as a difference equation

$$\underline{x}\,(t + \Delta t) = \underline{x}(t) + \underline{f}\,(\underline{x},\ \underline{z},\ t)\,\Delta t \tag{2}$$

The return function J is given by

$$J = \int_{t_o}^{t_f} h(\underline{x}, \underline{z}, s)\, ds \qquad (3)$$

where h = profit (or cost) function

s = dummy variable for integration

t_o, t_f = initial and final time respectively

Define $I(\underline{x}, t)$ = maximum profit for a process with initial state $\underline{x}$ and initial time t.

Then
$$I(\underline{x}, t) = \max_{\substack{\underline{z}(s)\in Z \\ t \le s \le t_f}} \int_{t}^{t_f} h(\underline{x}(s), \underline{z}(s), s)\, ds \qquad (4)$$

By the principle of optimality [4]

$$I(\underline{x},t) = \max_{\underline{z}\in Z} \left[h(\underline{x},\underline{z},t)\Delta t + I(\underline{x} + f(\underline{x},\underline{z},t)\Delta t, t+\Delta t) \right] \qquad (5)$$

Eqn. (5) gives the necessary reccurence relation.

COMPUTATIONAL PROCEDURE

As in Section 2, the range of vector $\underline{x}$ and its quantization levels are taken to be

$$\bar{\beta_i} \le x_i \le \beta_i^{+} \qquad i = 1,2,\ldots\ldots n \qquad (6)$$

$$\text{and} \quad x_i = \bar{\beta_i} + j_i \Delta x_i \qquad j_i = 0,1,\ldots\ldots,N_i$$

where $N_i \Delta x_i = \beta_i^+ - \beta_i^-$

The control vector $\underline{z}$ could be similarly quantized but it is sufficient to assume that the set of all admissible control variables is given by

$$Z = \left\{ \underline{z}^{(1)}, \underline{z}^{(2)}, \ldots\ldots, \underline{z}^{(K)} \right\} \tag{7}$$

Next the entire state time space X-t is partitioned into several blocks. Each block covers w_i increments along the x_i axis and ΔT along the t axis. Each block is indexed by its maximum value of $\underline{x}$ and t; thus

$$B (j_1, j_2, \ldots\ldots, j_n, m) = \left\{ \underline{x}, t \right\} \tag{8}$$

such that

$$(j_i - 1) w_i \Delta x_i \leq x_i \leq j_i w_i \Delta x_i$$

$$(m-1)\Delta T \leq t - t_o \leq m T$$

where $j_i = 1, 2, \ldots\ldots, J_i$

$$m = 1, 2, \ldots\ldots, M$$

$$J_i w_i \Delta x_i = \beta_i^+ - \beta_i^- \qquad i = 1, 2, \ldots\ldots n$$

and $$M\Delta T = t_f - t_o$$

The value of w_i is small, usually between 1 and 5.

For a two-dimensional problem, the blocks are three-dimensional rectangular solids. In Fig. 2 , a few blocks for a hypothetical two-dimensional model are illustrated.

Within the block, the time interval ΔT is further divided into smaller intervals Δt. The value of Δt is of the order used

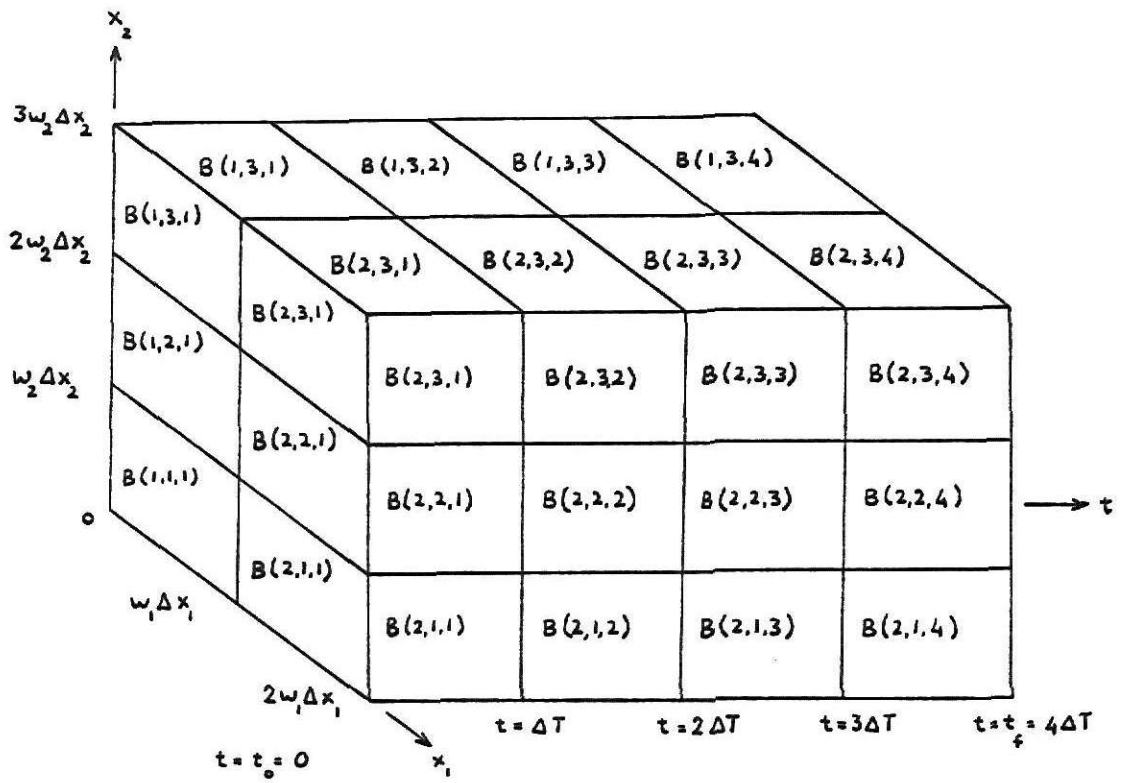


Fig. 2. Blocks for two-dimensional problem, $B(x_1, x_2, t)$.

in the conventional method. The quantized value of time t is thus given by

$$t = t_o + (m - 1) \Delta T + s \Delta t, \quad s=0,1,.....,S \qquad (9)$$

where $S\Delta t = \Delta T$

The value of S is usually between 5 and 15.

To compute the optimal control at a given $\underline{x}$ and t, each of the admissible control $\underline{z}^{(k)} \in Z$ is applied. For each such control, the minimum time interval $t^{(k)}$ required to change at least one of the state variables by one increment is given by

$$\delta t^{(k)} = \min_{i=1,2,..,n} \left\{ \left| \frac{x_i}{f_i(\underline{x}, \underline{z}^{(k)}, t)} \right| \right\} \qquad (10)$$

The corresponding next state is computed as

$$\underline{x}^{(k)} = \underline{x} + \underline{f} (\underline{x}, \underline{z}^{(k)}, t) \delta t^{(k)} \qquad (11)$$

It is noted that although the increment Δx_i is fixed, the rate of change $\underline{f} (\underline{x}, \underline{z}, t)$ can vary over a wide range and therefore, $\delta t^{(k)}$ can take on any continuous value.

The optimal profit at this state and time $I(\underline{x}^{(k)}, t+\delta t^{(k)})$ is computed by interpolation using the previously computed values at quantized states and at times $t+\Delta t$ and $t+2\Delta t$. If a control $\underline{z}^{(k)}$ is such that none of the state variables change, then $\underline{x}^{(k)} = \underline{x}$ and $t^{(k)}$ is set equal to Δt.

Next, the profit for each control over the interval $\delta t^{(k)}$ is found by evaluating $h (\underline{x}, \underline{z}^{(k)}, t)$. The optimal profit at $\underline{x}$ and t are then calculated from the principle of optimality.

$$I(\underline{x},t) = \max_{k=1,2,\ldots,K} \left\{ h(\underline{x}, \underline{z}^{(k)}, t)\delta t^{(k)} + I(\underline{x}^{(k)}, t+\delta t^{(k)}) \right\} \tag{12}$$

The value $\underline{z}^{(k)}$ which gives the maximum profit is clearly the optimal control at state $\underline{x}$ and time t.

The computation of optimal profit for a one-dimensional problem at $x = 7\Delta x$ and $t = 19\Delta t$ is illustrated in Fig. 3. For simplicity it is assumed that there are five admissible controls given by

$$Z = \left\{ z^{(1)}, z^{(2)}, z^{(3)}, z^{(4)}, z^{(5)} \right\}$$

The next state for each of the above controls is given by

$$\begin{aligned} x^{(k)} &= x\,(\,19\Delta t + \delta t^{(k)}\,) \\ &= 7\Delta x + \Delta x = 8\Delta x, \qquad k = 1,2 \\ &= 7\Delta x - \Delta x = 6\Delta x, \qquad k = 4,5 \\ &= 7\Delta x, \qquad\qquad\qquad k = 3 \end{aligned}$$

Since control $z^{(3)}$ is such that the next state $x^{(k)}$ does not change at all, $x^{(k)}$ is set equal to x.

The optimal profit at each of the next states is computed using interpolation in time based on values at the existing states. The known values at the grid points are shown by small circles in the figure. Two linear interpolation formulas, one for $x = 8\Delta x$ and the other for $x = 6\Delta x$ are defined as

$$I(8\Delta x,t) = I_{a1} + \frac{I_{a2} - I_{a1}}{t}(\,t - 20\Delta t\,) \tag{13}$$

$$I(6\Delta x,t) = I_{c1} + \frac{I_{c2} - I_{c1}}{t}(\,t - 20\Delta t\,) \tag{14}$$

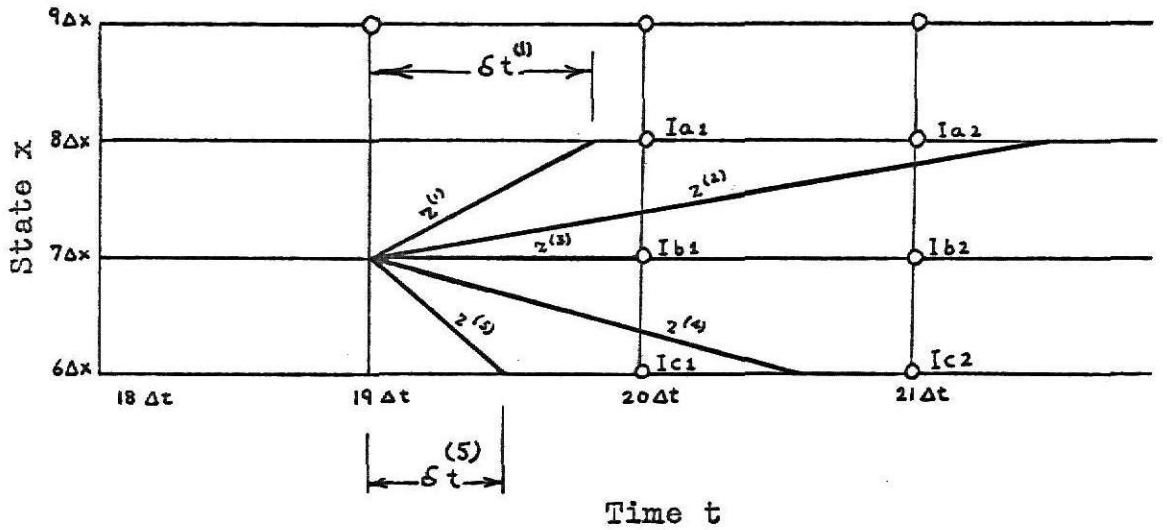


Fig. 3. Processing within a block

Depending on control $z^{(k)}$, the profit function now becomes

$$J(z^{(k)}) = h(7\Delta x,\ z^{(k)},\ 19\Delta t)\delta t^{(k)} + I(8\Delta x,\ 19\ t+\delta t^{(k)}) \qquad k=1,2$$

$$= h(7\Delta x,\ z^{(k)},\ 19\Delta t)\delta t^{(k)} + I(6\Delta x,\ 19\ t+\delta t^{(k)}) \qquad k=4,5$$

$$= h(7\Delta x,\ z^{(k)},\ 19\Delta t)\Delta t + I_{b1}, \quad k=3 \qquad (15)$$

The optimal profit is the maximum of the five quantities $J(z^{(k)})$ and the optimal control is that control for which the maximum is attained.

This procedure is repeated for other states in the block at time $t = 19\Delta t$. When all the points at time $t = 19\Delta t$ have been computed, the optimal points at $t = 21\Delta t$ are printed out from the fast memory and the optimal points at $t = 18\Delta t$ are computed on the basis of optimal points at $t = 19\Delta t$ and $t=20\Delta t$. This process is continued for all states in the block at time t given by

$$t = (\ m-1\ )\Delta T + s\Delta t, \qquad s= 0,1,.....,(S-1)$$

To process an adjacent block, the sequence of optimal points computed along the boundary between two blocks are used as the starting points for the next block. In order to determine the order in which the blocks are processed, the direction in which optimal trajectories are most likely to move must be determined first. This direction is called the "preferred direction of motion". The order of processing should be in this

preferred direction. In the production planning problems to be discussed later in this report, this knowledge about the preferred direction of motion is effectively exploited to determine the order of processing of blocks.

For a detailed treatment of state increment dynamic programming, the reader is referred to Larson [8,9].

COMPARISON WITH CONVENTIONAL DYNAMIC PROGRAMMING

The basic difference between the new method and conventional dynamic programming lies in the choice of the time interval over which control is applied. In conventional dynamic programming, a fixed time interval Δt is used whereas in the state increment method, this time interval is the minimum time needed for at least one of the state variables to change by one increment. The number of fast memory locations needed in conventional dynamic programming is proportional to the total number of distinct states. In Section 3 it was stated that this total number N for an n-dimensional problem is given by

$$N = \prod_{i=1}^{n} N_i \tag{16}$$

where N_i = number of quantization levels of the ith state variable.

On the other hand, the number of locations required by the new method is proportional to the number of distinct states within a block. If each block covers w_i increments along the ith state variable, then the number of such distinct states is given by

$$N' = \prod_{i=1}^{n} (w_i + 1) \qquad (17)$$

From equations (16) and (17), it is clear that the factor N/N' increases rapidly with the number of state variables. With three state variables and $N_i = 100$, N equals 100^3 or 10^6. During computation, at least two memory locations for each state are needed. One location is necessary to store the optimal profit for the previous stage, that is, at $t=t_o+(m-1)\Delta t$. The second location is required to store the value of the optimal profit being computed at the current stage, that is, at $t= t_o + m\Delta t$. Therefore, the fast memory requirement is actually 2×10^6.

For the same problem to be treated by the state increment method, the number of locations required is simply $2N'$. If in addition, points on the boundaries of the previously computed block are stored in the fast memory and if there are S such optimal points per state in an interval Δt, then the fast memory requirement becomes

$$N' = 2 \prod_{i=1}^{n} w_i + S(\prod_{i=1}^{n} (w_i+1) - \prod_{i=1}^{n} w_i) \qquad (18)$$

If the block size is chosen so that $w_i = 2$ for i=1,2,3 and S = 5, then N =111. If $w_i = 3$ and s = 10, which is rather a large set of values, then n = 424, still far less than $2\ 10^6$ locations required in conventional dynamic programming. This factor of reduction increases exponentially with the number of state variables.

In terms of computing time, the new method compares well with conventional dynamic programming. With certain sophisticated extensions of the new method which are beyond the scope of this report, Larson [8,9] has shown that some reduction in computationtime is sometimes possible. There is seldom any significant increase in the computaion time even after allowing for the additional complexity required in writing the computer programs.

The accuracy of the results obtained by the new method is very nearly the same as that of conventional dynamic programming provided that the increment sizes in time, state and control variables and the interpolation formulas used are the same. An additional condition to obtain good accuracy is that the time interval δt over which control is applied must be about the same order of magnitude as Δt, the fixed time interval into which a block is partitioned along the t axis. As will be seen in the production planning problems to be discussed later in this report, this condition cannot be always satisfied. Thus further investigation and improvement in this area are needed

Some of the important disadvantages of state increment dynamic programming can be summarized as follows. First, discrete processes cannot be treated by the new method. Second, there must exist enough feeling for the general behaviour of the optimal trajectories so that a preferred direction of motion can be defined to facilitate the processing of the blocks in proper sequence. If the preferred direction is unknown, compu-

tation time can be increased considerably. Third, if a problem is of such a nature that after applying control for a small time Δt, the next state is not in a neighborhood of the state at which control was applied, then state increment dynamic programming cannot be applied effectively. Fortunately, however, in most optimization problems of physical significance, the above-mentioned difficulties do not constitute a severe limitation of the method.

In general it can be concluded that for a large class of problems, the new method retains the many good features of dynamic programming while it greatly reduces the fast memory requirements. Thus, for a large number of problems, the size of the fast memory is no longer the limiting factor.

5. APPLICATIONS

In this section three hypothetical models are considered. All the three models represent production planning and scheduling situations. The first model involves a discrete case of production scheduling under a learning effect. Because of its discrete nature, this model cannot be handled by state increment approach.

The second and third models involve continuous cases. The second model is a simple one and has only one state variable. The third model is, however, more complex and has two state variables. Solutions from both of these models are then derived by applying first conventional dynamic programming and then the state increment method. A comparison is then made in each case to highlight the differences as well as the similarities between the two methods.

A PRODUCTION SCHEDULING MODEL WITH LEARNING EFFECT

This model is taken from a paper by Spradlin and Pierce [16]. The basic problem is to obtain an optimal production schedule for a product produced in lots and whose production rate can be described by a learning curve. A firm gets an order for the production of a fixed number of units. The firm can produce the entire order in one lot or in several lots. The deliveries, however, will be accepted only at a contracted demand rate. If only one lot is produced, production will remain uninterrupted

but inventory will be high. On the other hand, if the same order is produced in several lots, inventory cost is reduced, but set-up costs will increase; furthermore, as production is interrupted several times, there will be an additional penalty due to the loss in learning effect.

Let X = total number of units to be produced

K = labor cost of producing the first unit

S = set-up cost per order

C = inventory carry charges per uint per unit of time

M = minimum production lot size

d = demand rate

z_i = the ith production lot size, i=1,2,.....,n.

The equation for the learning curve is given by

$$K_x = Kx^m$$

where K_x = labor cost to produce the xth unit

$m = \log N / \log 2$

$N = \dfrac{\text{labor cost of producing jth unit}}{\text{labor cost of producing 2jth unit}}$

A ratio of N = 0.80 is commonly referred as a 80% learning curve. It means that if it takes 100 hours to produce the first unit, it will take 80 hours for the second unit and 64 hours for the fourth unit and so on. For a greater explanation of equation (1), the reader is referred to Spradlin and Pierce [16].

Define $g_n(x, z_n)$ = the cost of producing and storing z_n more units given that x units have been produced in (n-1) lots.

and $f_n(x)$ = the cost of an optimal production schedule for n lots consisting of a total of x units.

Then by the principality of optimality,

$$f_n(x) = \min_{z_n} \left\{ g_n(x-z_n, z_n) + f_{n-1}(x-z_n) \right\} \qquad (2)$$

which is the required functional equation with the necessary reccurence relationship.

The labor cost for a lot of size z_n when it is begun as the xth unit on the learning curve is equal to the area under the curve $K.x^m$ from $x-0.5$ to $x-z_n-0.5$ or

$$\int_{x-0.5}^{x+z_n-0.5} K.s^m.ds$$

where s = dummy variable for integration

The storage costs for a lot of size z_n from the time it is produced untill it is depleted are

$$C.\frac{z_n}{d} .\frac{z_n}{2} = \frac{C}{2d}(z_n)^2$$

The loss in the learning rate is accounted for by moving the integral representing the labor cost back in a cumulative manner, M units toward the origin each time production is interrupted. Thus

$$g_n(x-z_n,z_n) = S + \frac{C}{2d}.z_n^2 + \int_{x-z_n-(n-1)M+0.5}^{x-(n-1)M+0.5} K.s^m.ds \qquad (3)$$

Using equation (3) in equation (2), $f_n(x)$ can now be evaluated for all $n \geqslant 1$.

Figure 4 gives a simplified flowchart of the computational procedure. The numerical value assumed are

X = 100	m = - 0.234 for learning effect
K = 100	= 0 for no learning effect
M = 10	C = 20
S = 625	d = 10

The computer program in FORTRAN and a sample printout of the results obtained are given in Appendix B. The optimal policies are tabulated in Table 1.

In this problem, there are actually two control variables, namely, the number of lots and their respective lot sizes. The first variable has the same significance as the number of stages in other problems formulated by dynamic programming.

It is of interest to note that for the case m = 0, i.e., when the production was not subject to any learning effect, the optimal production schedule is the same as given by the more common square-root formula. The results obtained are comparable to those obtained by Spradlin and Pierce [16].

It can be seen from equation (3) that the set-up costs affect the solution in no other way than in the number of lots. In other words, suppose that the set-up costs are unchanged but the number of lots is the same; then the breakdown into lots will depend only on the other costs and the learning curve.

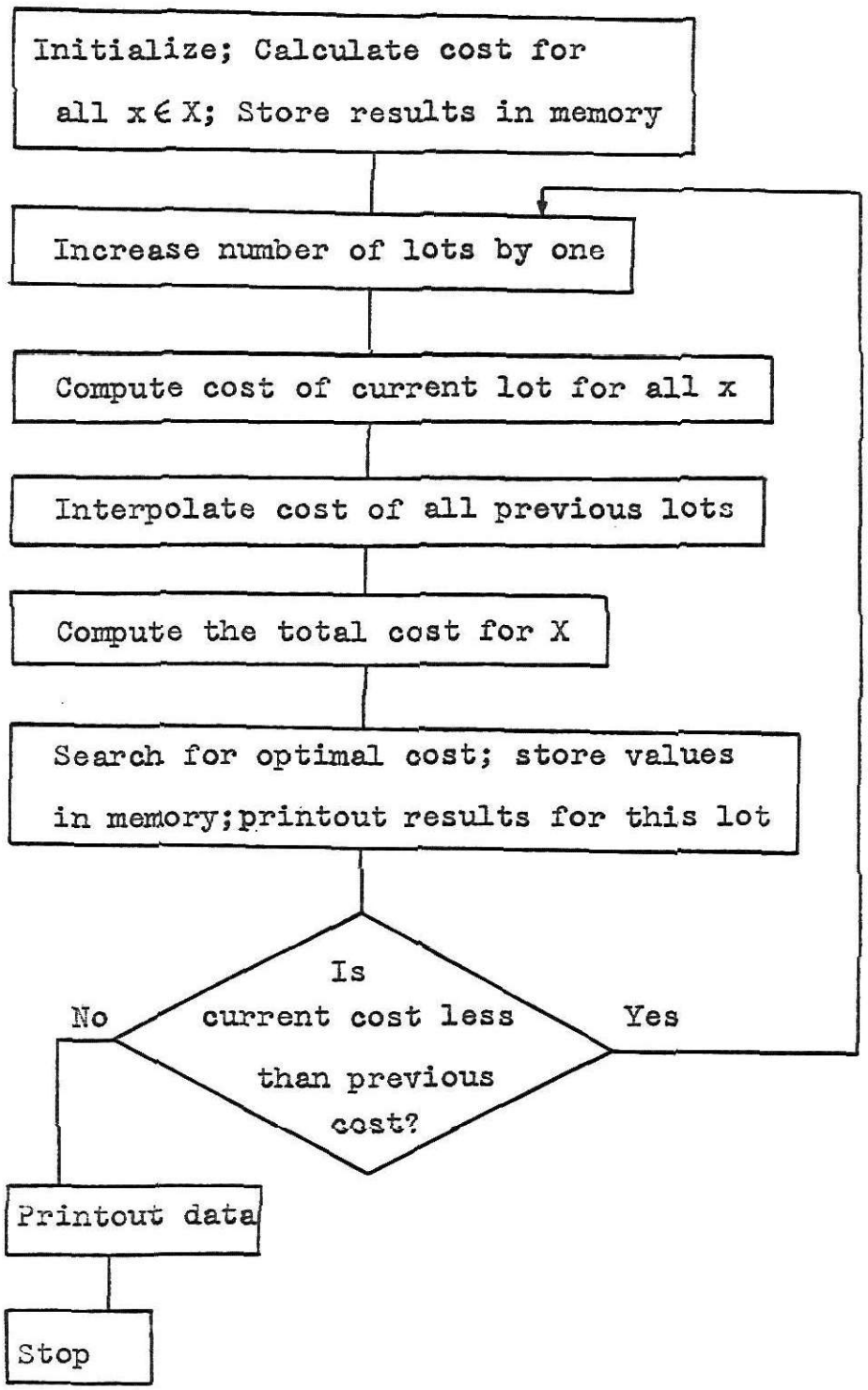


Fig. 4. Flowchart for the production scheduling model with learning effect.

Table 1. Results of production scheduling problem with learning effect.

	with learning effect m = -0.234	without learning effect, m=0
Optimal cost	9,766.3	15,000.0
Optimal number of lots	4	4
Optimal lot sizes (z_1, z_2, z_3, z_4)	26, 25, 25, 24	25, 25, 25, 25

AN INVENTORY MODEL WITH ONE STATE VARIABLE

This model is taken from an unpublished paper by Lee and Shaikh [12]. Consider the case of a manufacturing organization whose sales rate is known with certainty. The rate of change of inventory level I(t) is given by

$$dI(t)/dt = P(t) - Q(t) \tag{4}$$

where $P(t)$ = production rate at time t

$Q(t)$ = sales rate at time t

The management wishes to minimize the cost functional given by

$$C_T = \int_0^T \left[C_I \, (I_m - I(t))^2 + C_P \exp(P_m - P(t))^2 \right] dt \tag{5}$$

where C_T is the total cost of inventory and production. C_P is the minimum production cost which occurs when the production rate equals P_m. The quantity P_m can be considered as the capacity of the manufacturing plant. Since the plant is designed for a capacity P_m, an increase in capacity may require additional equipments and manpower and this can be very expensive. On the other hand, a decrease in capacity will be equally expensive due to the maintenance of unused equipments and manpower which due to contract agreements cannot be reduced easily. The quantity I_m can be considered as the capacity for storage of inventory. C_I is the inventory carrying cost. In many practical situations, the minimum storage cost is obtained when the storage capacity is completely filled. Furthermore, the cost functional has the smoothing capability which is frequently desirable for many manu-

facturing processes. In this case I_m and P_m can be considered as the desirable inventory and production levels. It is further assumed that the sales forecast is known and is given by the linear relation

$$Q(t) = a + bt \tag{6}$$

The initial inventory is given as

$$I(0) = c \tag{7}$$

This above production planning problem can be formulated as a dynamic programming model. Let,

$$x(t) = I(t) \tag{8}$$

$$z(t) = P(t) \tag{9}$$

Thus eqn. (4) becomes

$$\begin{aligned} dx(t)/dt &= z(t) - Q(t) \\ &= z(t) - a - bt \end{aligned} \tag{10}$$

From eqn. (10) we get the difference equation

$$x(t+\Delta t) = x(t) + (z(t)-a-bt)\Delta t \tag{11}$$

To obtain the cost, we need to compute the integral in eqn. (5) over the limits $n\Delta t$ to $(n+1)\Delta t$. If t is small, this integral can be approximated by

$$\begin{aligned} &\int_{n\Delta t}^{(n+1)\Delta t} \left[C_I(I_m-x(t))^2 + C_P \exp(P_m-P(t))^2\right] dt \\ &= \left[C_I(I_m-x(t))^2 + C_P \exp(P_m-P(t))^2\right]\Delta t \end{aligned} \tag{12}$$

The numerical values assumed are:

$a = 2$ $\quad b = 1$ $\quad c = 5$

$C_I = 0.1$ $\quad I_m = 10$ $\quad P_m = 5$

$C_p = 0.001$ $t_o = 0$ $t_f = 1$

a. Solution by conventional dynamic programming

A solution is first derived by straightforward application of conventional dynamic programming. The number of stages into which the entire process can be divided is determined by the accuracy required and the cost of computation involved. If the number of stages is too small, the solution obtained will be very approximate. On the other hand, if the number of stages is too large, the computation time or the cost is increased. For this study, the number of stagrs considered are 5, 10 and 20.

The range of state variable x and control variable z is taken to be between 0 and 10. (Clearly x and z cannot be negative).

The control variable z is quantized into 100 increments of step size = 0.1.

The value of x is taken to be as follows:

number of stages	t	Δx
5	0.20	1.00
10	0.10	0.50
20	0.05	0.25

The reason for taking different values of x will become clear when we obtain the solution by state increment method. Since we wish to compare the two methods, we take the same value of Δx which we will use later on in the state increment method.

This problem was solved on an IBM 360 computer. The flow-

chart of the computer program is given in Fig. 19 in Appendix A. The computer program was written in FORTRAN language and is given together with a sample prinout for the 10-stage process in Appendix B. The results for the particular given initial condition were interpolated manually and are tabulated in Tables 2-4. The same results are also plotted in Figs. 5-10.

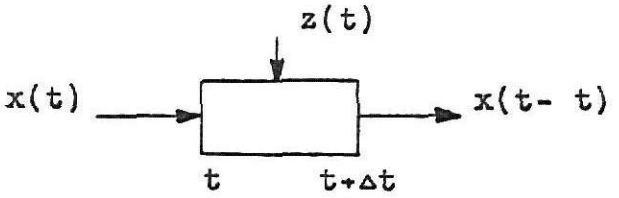
As will be seen from Tables 2-4, the value of the optimal cost is 0.925 for 5- and 10-stage process and 0.922 for 20-stage process. There is no significant difference between the two values. The reason is that the inventory and the cost function are smmoth functions as is also evident from Figs. 9 and 10. We therefore, conclude that for this particular model, increasing the number of stages does not improve the value of optimal cost significantly. This is, however, not the general case as will be seen in the solution of the second model to be discussed later.

This problem has been solved by Lee and Shaikh [12] by using gradient technique and the value of optimal cost obtained by them was 0.943. Thus, there is a difference of about 0.021 or 2.2% which is reasonable. The reason for this could be that the step sizes Δx and Δz are not sufficiently small.

b. Solution by state increment dynamic programming

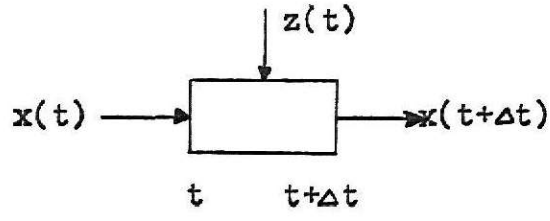
Next, a solution of the same model is obtained by using state increment dynamic programming. From the earlier solution we know that the state variable x always increases at each stage. This, then, is the preferred direction of motion. This fact is used in determining the computational scheme. Since the

Table 2. 5-stage process by conventional dynamic programming.



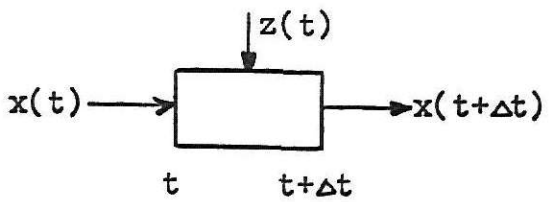
t - t+Δt	x(t)	z(t)	x(t+Δt)	C_T	Q(t)
0.0 - 0.2	5.00	7.10	6.00	.422	2.10
0.2 - 0.4	6.00	7.10	6.96	.686	2.30
0.4 - 0.6	6.96	7.00	7.86	.832	2.50
0.6 - 0.8	7.86	6.81	8.68	.900	2.70
0.8 - 1.0	8.68	6.31	9.38	.926	2.90

Table 3. 10-stage process by conventional dynamic programming.



t - t+Δt	x(t)	z(t)	x(t+Δt)	C_T	Q(t)
0.0-0.1	5.00	7.20	5.52	.235	2.05
0.1-0.2	5.52	7.10	6.01	.426	2.15
0.2-0.3	6.01	7.10	6.49	.574	2.25
0.3-0.4	6.49	7.00	6.96	.691	2.35
0.4-0.5	6.96	7.00	7.41	.772	2.45
0.5-0.6	7.41	6.92	7.85	.834	2.55
0.6-0.7	7.85	6.83	8.30	.878	2.65
0.7-0.8	8.30	6.64	8.69	.902	2.75
0.8-0.9	8.69	6.52	9.05	.919	2.85
0.9-1.0	9.05	6.08	9.35	.925	2.95

Table 4. 20-stage process by conventional dynamic programming.



t - t+Δt	x(t)	z(t)	x(t+Δt)	C_T	Q(t)
.00 - .05	5.00	7.20	5.26	.120	2.02
.05 - .10	5.26	7.10	5.50	.231	2.07
.10 - .15	5.50	7.10	5.75	.331	2.13
.15 - .20	5.75	7.10	6.00	.422	2.17
.20 - .25	6.00	7.10	6.24	.503	2.22
.25 - .30	6.24	7.10	6.49	.573	2.27
.30 - .35	6.49	7.00	6.72	.632	2.32
.35 - .40	6.72	7.00	6.95	.685	2.38
.40 - .45	6.95	7.00	7.18	.731	2.42
.45 - .50	7.18	6.93	7.40	.770	2.47
.50 - .55	7.40	6.90	7.62	.804	2.52
.55 - .60	7.62	6.90	7.84	.832	2.57
.60 - .65	7.84	6.80	8.05	.854	2.63
.65 - .70	8.05	6.78	8.25	.873	2.67
.70 - .75	8.25	6.70	8.45	.888	2.72
.75 - .80	8.45	6.62	8.64	.899	2.77
.80 - .85	8.64	6.55	8.82	.907	2.82
.85 - .90	8.82	6.47	9.00	.914	2.88
.90 - .95	9.00	6.30	9.17	.919	2.92
.95 - 1.00	9.17	5.84	9.32	.922	2.97

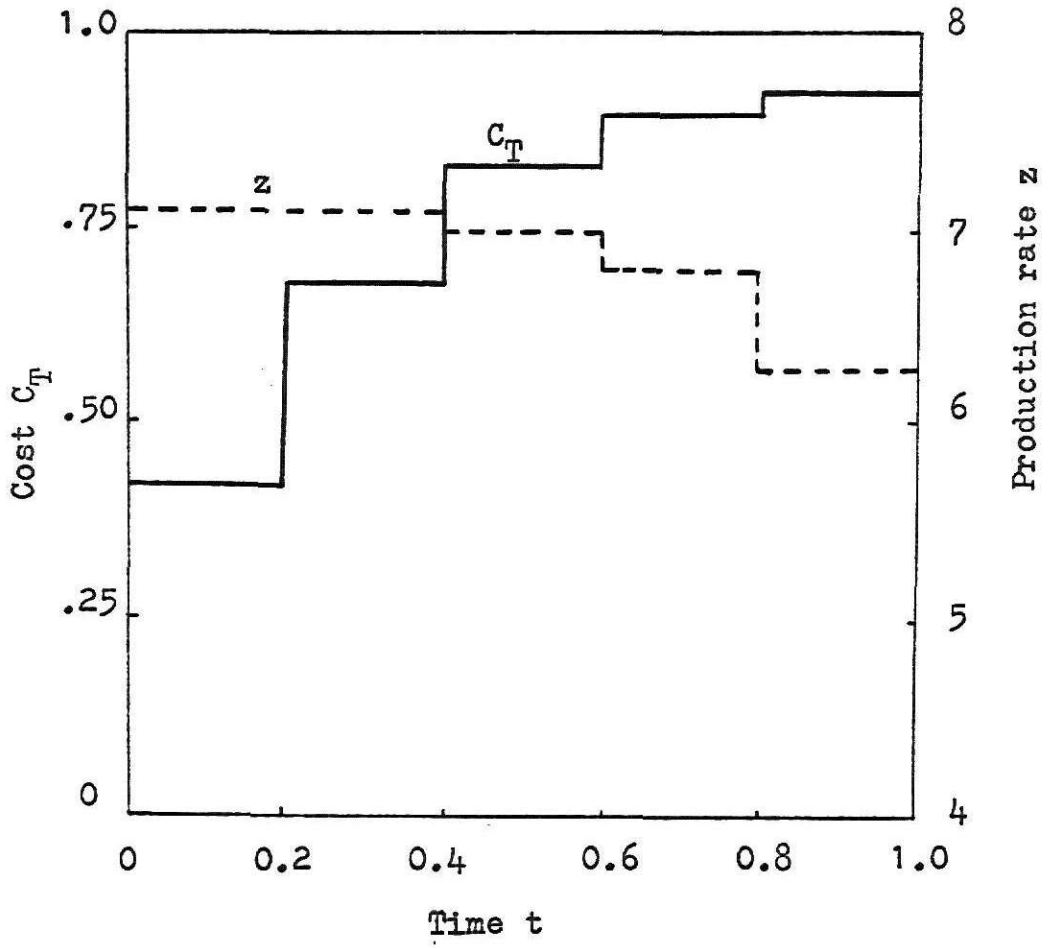


Fig. 5. Cost and production rate for 5-stage process

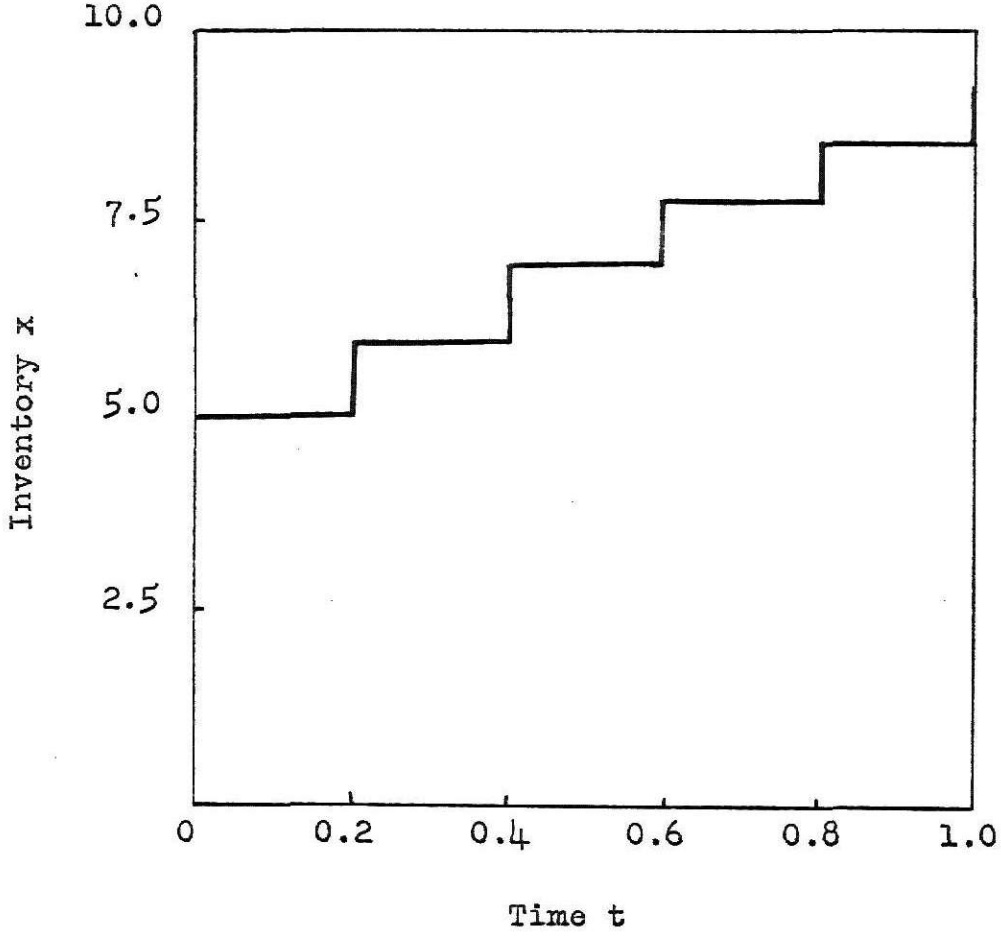


Fig. 6. Inventory for 5-stage process.

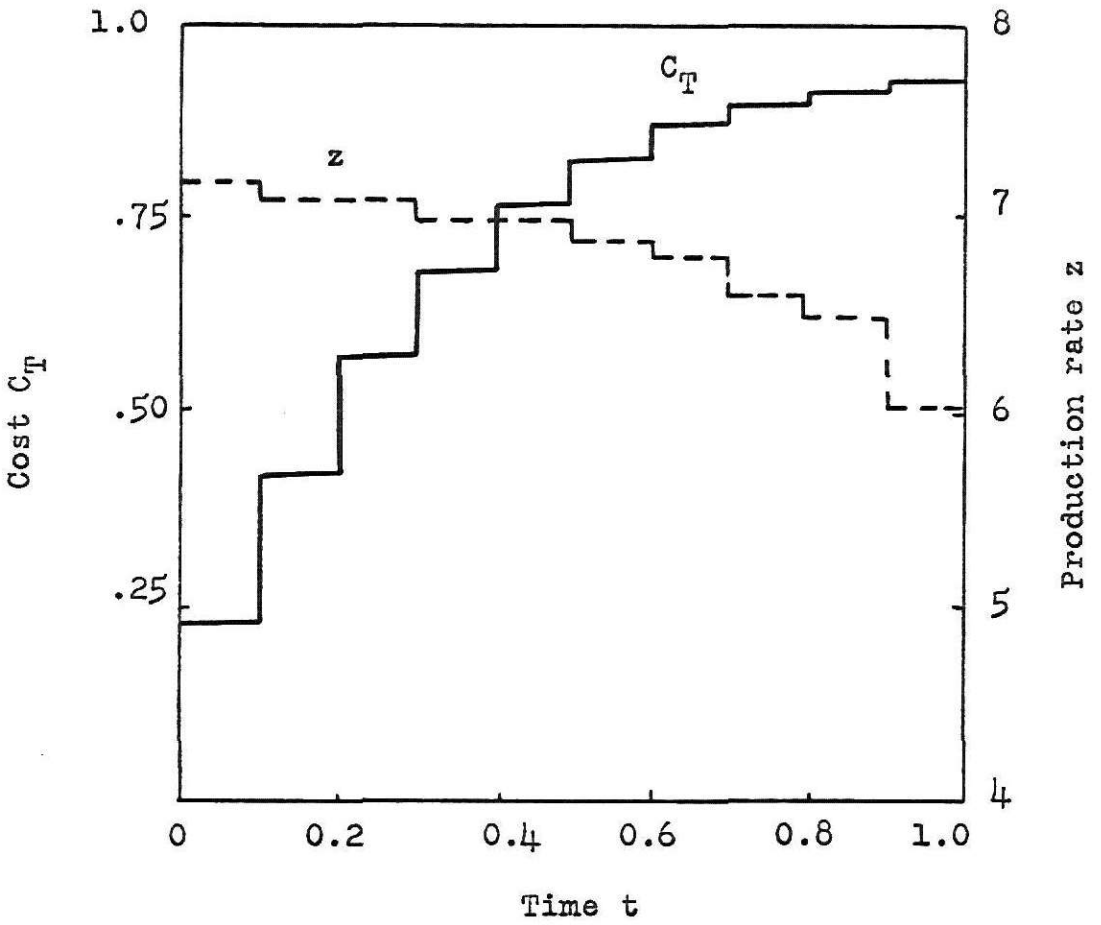


Fig. 7. Cost and production rate for 10-stage process.

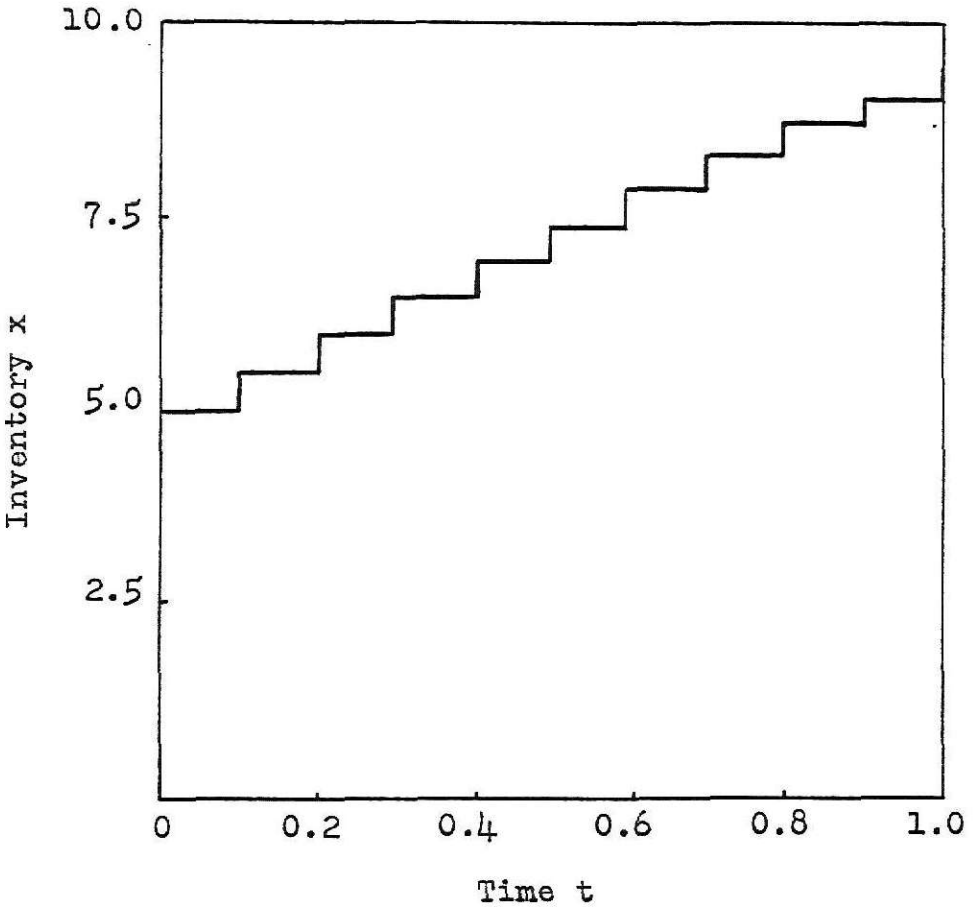


Fig. 8. Inventory for 10-stage process.

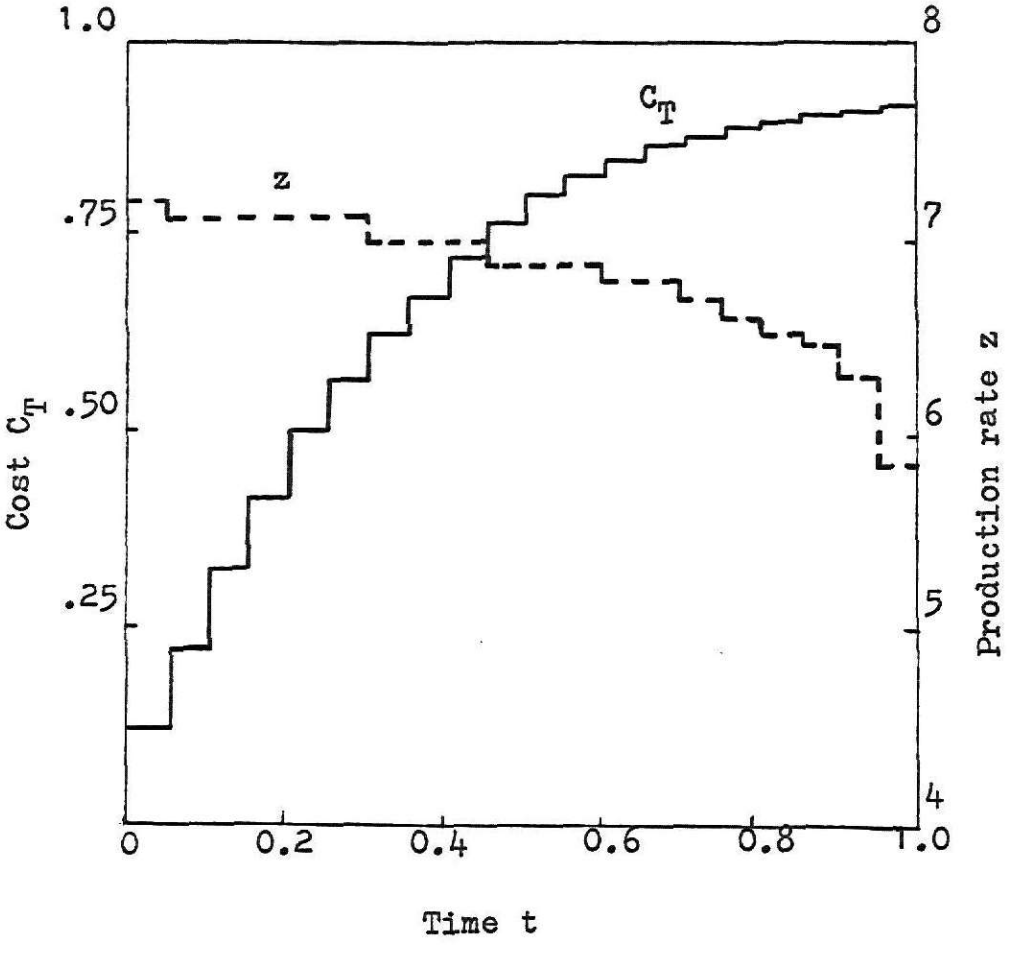


Fig. 9. Cost and production rate for 20-stage process.

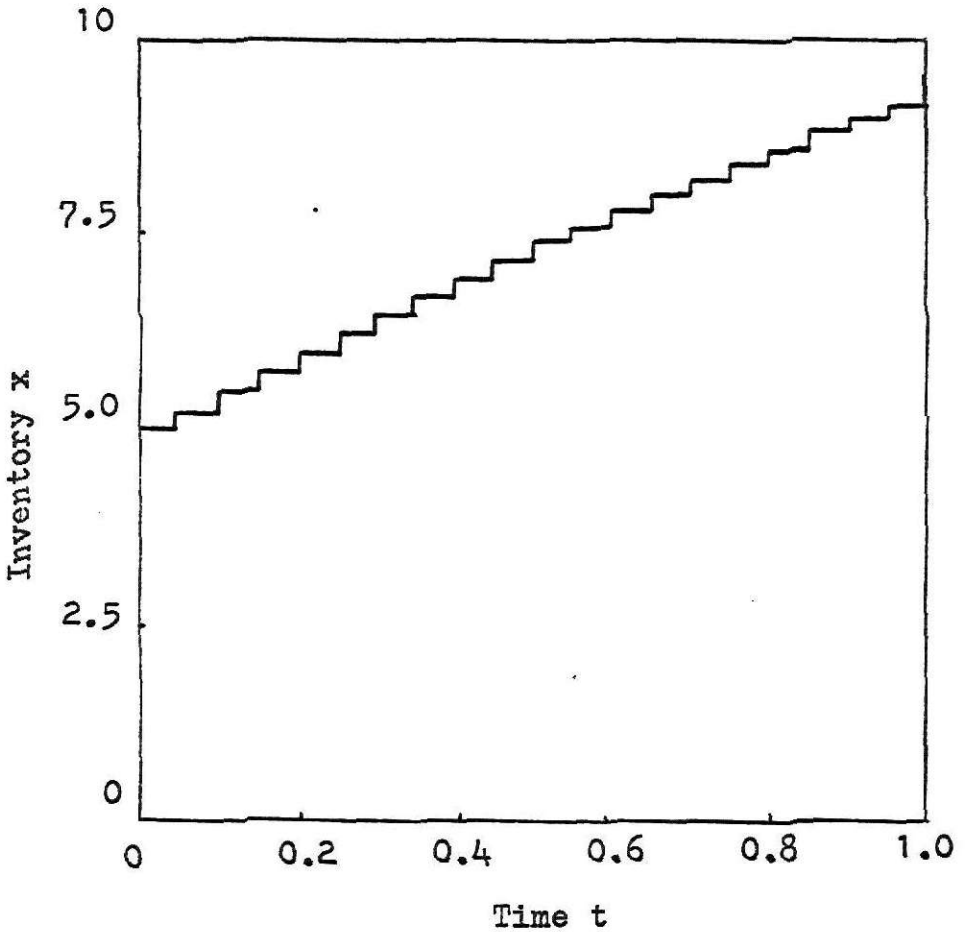


Fig. 10. Inventory for 20-stage process.

maximum number of stages is 20 and also since the problem is well behaved, only one time block for the entire interval is used. Thus

$$M\Delta t = \Delta T = t_f - t_o = 1,$$

where $M = 1$

The time interval T is further divided into smaller increments Δt. Thus

$$S\,\Delta t = \Delta T$$

The value of S is the same as the number of stages, i.e., 5,10 and 20.

To get good accuracy by state increment dynamic programming, the average value of the time interval δt required to change at least one of the states by one increment Δx should be about Δt. In other words, $\Delta x/\Delta t \approx \Delta x/\delta t \approx 1$. If Δx is too small or too large, the new method largely relies on interpolation, leading to inaccurate results. Such a condition when Δx is too small is illustrated in fig. 11a. Optimal profit at point P is to be computed using the known values of the optimal profit at points P_1 and P_2. Since Δx is too small, the state $n\Delta x$ is increased by one increment Δx to $(n+1)\Delta x$ when control z is applied for only a short duration δt. As a result, the point P_3 is far away from the points P_1 and P_2. Therefore, the value of the optimal profit determined at P_3 by extrapolation based on the values at P_1 and P_2 will be inaccurate.

From the earlier solution by conventional dynamic programming, the total change in x during the entire time period ($t_f - t_o$) is about 4.32, being the difference between the initial inventory of 5 and the final inventory of 9.32. Therefore, for a

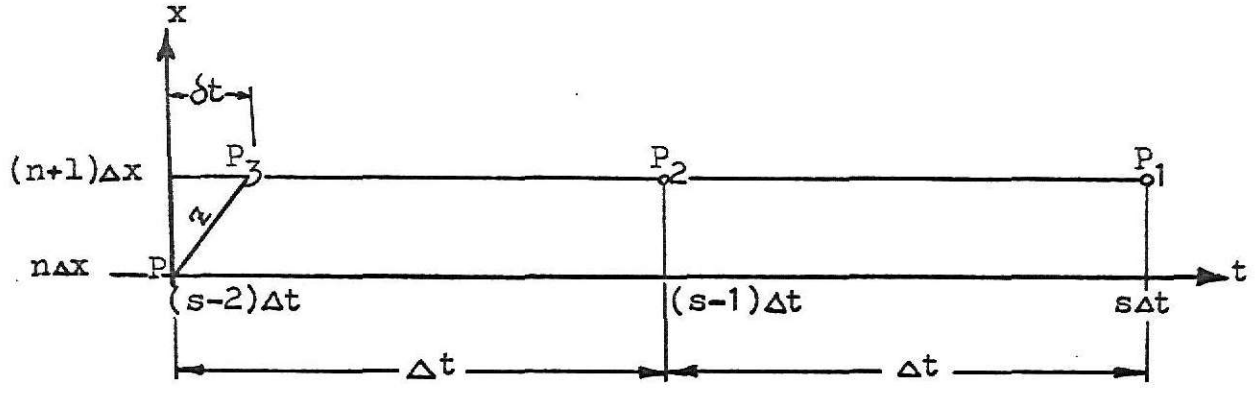


Fig. 11a. Processing with too small Δx

5-stage process with t=0.2, the value of x should be around 4.32 x 0.2 or say 1.0. The value of Δx for other values of Δt is similarly determined.

Since the state variable x always increases, the width $w\Delta x$ of the time block is taken to be equal to Δx. Thus w=1.

Figure 11 shows the computational scheme for the 5-stage process. Consider the block containing x=10 and x=9. To process this block the boundary values of the cost function at points A_1 through A_6 for x=10 must be known. The value at A_6 is by definition zero and the value at A_5 is found as in the conventional method. Since we do not know the values at the other points A_1-A_4, these are assumed to be zero arbitrarily. As a result of this assumption, the values of the cost function at points marked C will be incorrect. To overcome this difficulty, the range of x is increased to 15 instead of 10. In this way the cost function at all points for $0 \leq x \leq 10$ will be correct.

Since x increases for all values of x and t, we process the blocks containing higher values of x first and proceed towards the origin. Consider the block containing x=3 and x=2 in Fig. 11. The points D_1 through D_6 are on the common boundary of the blocks B (4, 1) and B (3, 1). These points are used to process the points E_1 through E_6 in the next block B (3, 1). The value of the optimal cost at E_6 is zero since this point lies at time $t=t_f$. The value at E_5 can be determined by the conventional dynamic programming method. Notice, however, that to process this point E_5, no interpolation of the optimal points from the previous stage is required because the value of the

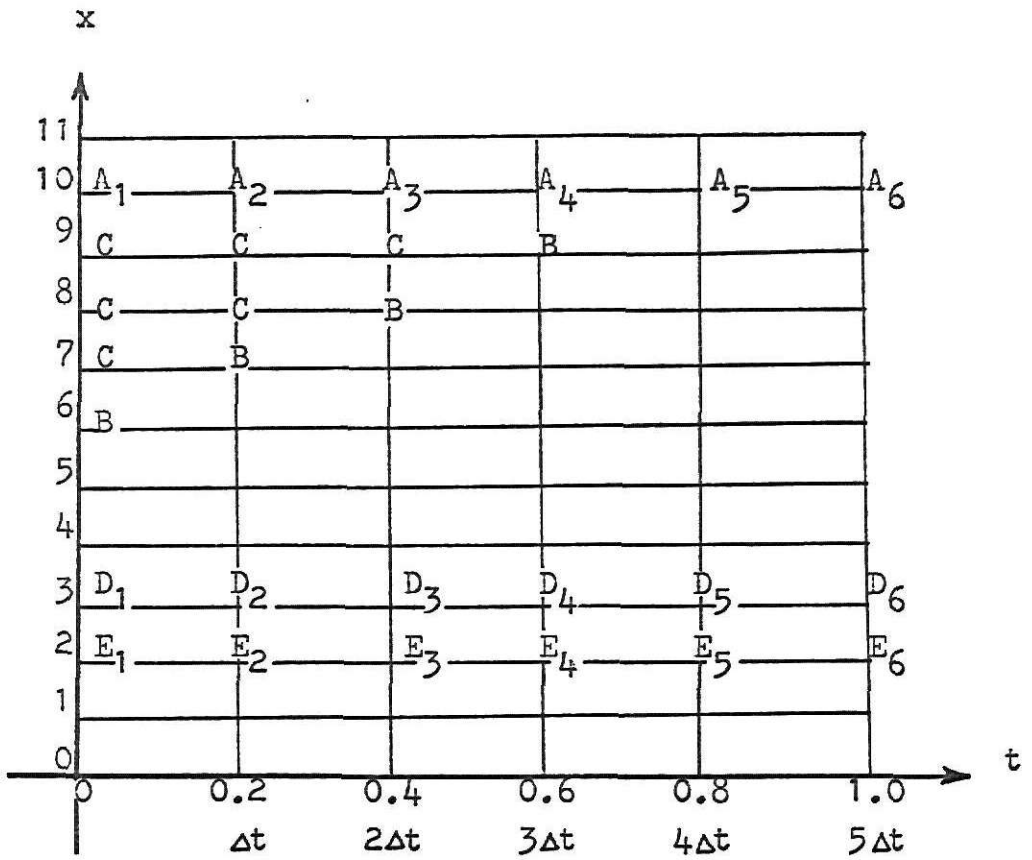


Fig. 11. Computation scheme for a 5-stage process.

optimal cost at the previous stage (that is at $t=t_f$) is zero for all values of x. This eliminates the necessity of storing any table in the computer.

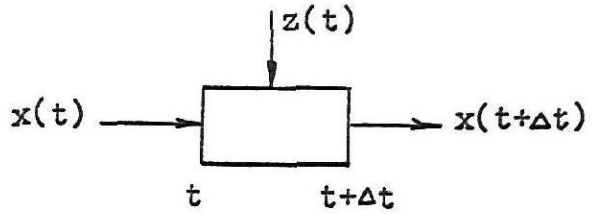
The point E_4 is processed by using the known values at D_6 and D_5. The procedure for processing point E_4, or in other words at x=2 x and t=3 t is similar to the procedure described in Section 4 and Fig. 3. However, in the present case, the interpolation formula given by equation (13), Section 4 is sufficiet for all controls since the value of x always increases.

The flowchart for the computer program is given in Appendix A, Fig. 20. The actual computer program in the FORTRAN language together with a sample of the computer printout of the results is given in Appendix B. The results for the given initial condition were interpolated manually and are given in Tables 5-7

c. Comparison

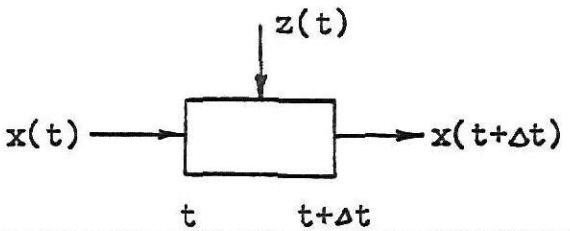
A comparison of the results obtained by two methods gives similar values. In fact, if the results obtained by state increment programming were to be plotted, the same graphs as given in Figs. 5-10 are obtained. The time taken in state increment programming is about 35% to 52% more. The number of fast memory locations required is,however, only half of what is needed in conventional dynamic programming. As will be seen in the second model, this reduction in the memory requirement is further enhanced when the number of state variables increases. The increase in the computation time is due to the fact that the range of x was increased.

Table 5. 5-stage process by state increment dynamic programming.



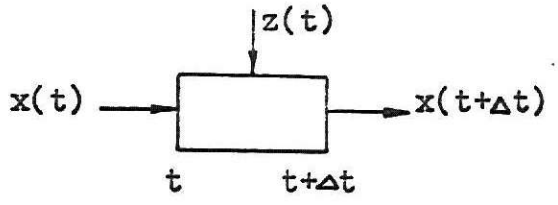
t - t+Δt	x(t)	z(t)	x(t+Δt)	C_T	Q(t)
.0 - .2	5.00	7.10	6.00	.422	2.10
.2 - .4	6.00	7.00	6.94	.686	2.30
.4 - .6	6.94	6.90	7.82	.818	2.50
.6 - .8	7.82	6.64	8.61	.882	2.70
.8 -1.0	8.61	6.42	9.32	.910	2.90

Table 6. 10-stage process by state increment dynamic programming.



t - t+Δt	x(t)	z(t)	x(t+Δt)	C_T	Q(t)
.0 - .1	5.00	7.20	5.52	.239	2.05
.1 - .2	5.52	7.10	5.99	.426	2.15
.2 - .3	5.99	7.10	6.48	.581	2.25
.3 - .4	6.48	7.00	6.97	.687	2.35
.4 - .5	6.97	6.90	7.44	.773	2.45
.5 - .6	7.44	6.90	7.87	.830	2.55
.6 - .7	7.87	6.75	8.27	.867	2.65
.7 - .8	8.27	6.65	8.65	.893	2.75
.8 - .9	8.65	6.53	9.00	.908	2.85
.9 - 1.0	9.00	6.10	9.31	.915	2.95

Table 7. 20-stage process by conventional dynamic programming.



t - t+Δt	x(t)	z(t)	x(t+Δt)	C_T	Q(t)
.00 - .05	5.00	7.20	5.26	.115	2.02
.05 - .10	5.26	7.20	5.52	.224	2.07
.10 - .15	5.52	7.10	5.77	.327	2.12
.15 - .20	5.77	7.10	6.02	.428	2.17
.20 - .25	6.02	7.10	6.26	.506	2.22
.25 - .30	6.26	7.10	6.50	.575	2.27
.30 - .35	6.50	7.00	6.73	.632	2.32
.35 - .40	6.73	7.00	6.96	.683	2.37
.40 - .45	6.96	7.00	7.19	.730	2.42
.45 - .50	7.19	6.93	7.41	.769	2.47
.50 - .55	7.41	6.90	7.61	.803	2.52
.55 - .60	7.61	6.90	7.83	.826	2.57
.60 - .65	7.83	6.80	8.04	.849	2.62
.65 - .70	8.04	6.70	8.25	.870	2.67
.70 - .75	8.25	6.70	8.45	.884	2.72
.75 - .80	8.45	6.62	8.64	.894	2.77
.80 - .85	8.64	6.55	8.82	.902	2.82
.85 - .90	8.82	6.43	9.00	.910	2.87
.90 - .95	9.00	6.20	9.16	.914	2.92
.95 - 1.00	9.16	5.84	9.31	.917	2.97

A summary of the comparison between the two methods is given in table 8.

AN INVENTORY AND ADVERTISING MODEL WITH TWO STATE VARIABLES

This model is an extension of a model formulated by Teichroew [15]. Consider a marketing situation where only a certain number of possible customers possess certain information about a firm's product. Suppose that the total number of such persons remains constant and that the diffusion of information occurs only through personal contact. The number of contacts made by an average informed person in an arbitrary unit of time is given by a " contact coefficient ". In a contact, the contactee receives information if he does not already have it; if he already has it, the contact is wasted in so far as increasing the number of informed persons is concerned.

Let $K(0) = K_o$ = number of informed persons at time t_o

N = total number of persons

c = contact coefficient; the number of contacts made by one informed person per unit time

$K(t)$ = number of informed persons at time t

Then $K(t)/N$ = proportion of informed persons at time t

$1-K(t)/N$ = proportion of uninformed persons at time t

$c.K(t)dt$ = contacts made during a time interval dt

Clearly $dK(t) = c.K(t)dt(1-K(t)/N)$

Thus the equation governing the process is

Table 8. Comparison of the results.

Number of stages	Optimal production cost, C_T		Computation time, in minutes		Number of fast storage locations used	
	DP	SI	DP	SI	DP	SI
5	.925	.910	.65	.88	22	12
10	.925	.915	1.68	2.55	42	22
20	.922	.917	6.18	9.29	82	42

DP: By dynamic programming

SI: By state increment programming

$$\frac{dK(t)}{dt} = c.K(t)\left[1 - \frac{K(t)}{N}\right] \qquad (13)$$

Suppose next that the firm can influence the number of contacts by spending money on advertising. In particular, it can increase the number of contacts made by the informed persons (above the ones included in c) by an additional number A per unit of time. Eqn. (13) then becomes

$$\frac{dK(t)}{dt} = K(t)(c+A(t))(1- \frac{K(t)}{N}) \qquad (14)$$

If each successful contact results in the sale of n units of the firm's product and if Q(t) represents the sale at time t, then

$$Q(t) = n\ K(t)$$

Letting n=1 and substituting Q(t) for K(t) in eqn. (14), we get

$$\frac{dQ(t)}{dt} = Q(t)(c+A(t))(1- \frac{Q(t)}{N}) \qquad (15)$$

Next the rate of change of the firm's inventory is given by

$$\frac{dx(t)}{dt} = P(t) - Q(t) \qquad (16)$$

where P(t) = production rate at time t

The production rate is assumed to be a linear function given by

$$P(t) = a + bt \qquad (17)$$

where a and b are constants.

As will be seen later this assumption is made to simplify the model by avoiding a second control variable.

The firm's mangement wishes to maximize the profit

$$S_T = \int_0^T \left[F\, Q(t) - C_I(P_I - x(t))^2 - C_A\, A\, Q \right] dt \qquad (18)$$

where S_T is the total net profit. F is the revenue from sale of one unit of the product. C_I is the inventory carrying cost and has the same significance as in the earlier model. P_I can be considered as the capacity for the storage of inventory. C_A is the cost of advertising.

Equations (15) through (18) completely represent the system. It has two state variables x(t) and Q(t) and one control variable A(t). To treat this model by dynamic programming, the system equations are reduced to difference equations.

Equation (15) is reduced to

$$Q(t+\Delta t) = Q(t) + Q(t)\left[c+A(t)\right]\left[1- \frac{Q(t)}{N} \right]\Delta t \qquad (19)$$

If this equation is used as it is, $Q(t+\Delta t)$ can exceed N which is not possible theoretically. To avoid this difficulty, the term $(1- Q(t)/N)$ is replaced by $(1- Q(t+\Delta t)/N)$. In this way, $Q(t+\Delta t)$ can never exceed N. Making this change and re-arranging the terms

$$Q(t+\Delta t) = \frac{Q(t)\left[1+(c+A(t)\right]\Delta t}{1+(c+A(t))Q(t)\Delta t/N} \qquad (20)$$

Eqn. (16) is reduced to

$$x(t+\Delta t) = x(t) + (P(t) - Q(t))\Delta t \qquad (21)$$

To obtain the profit, we need to compute the integral in eqn. (18) over the limits $n\Delta t$ to $(n+1)\Delta t$. If Δt is small, this integral can be approximated by

$$\int_{n\Delta t}^{(n+1)\Delta t} \left[F\,Q(t) - C_I(P_I - x(t))^2 - C_A\,A\,Q \right] dt$$

$$= \left[F\,Q(t) - C_I(P_I - x(t))^2 - C_A\,A\,Q \right]\Delta t \qquad (22)$$

The initial conditions and numerical values used are:

$P_I = 50$		$b = 100$	
$C_I = 0.15$		$c = 2$	
$C_A = 1.5$		$F = 10$	
$N = 150$		$x(0) = 20$	
$a = 70$		$Q(0) = 20$	$T = 1$

In addition, due to limited funds for advertising, the firm's policy is that the amount of advertising A(t) at any time should not exceed the value of 6. This, then, is a constraint on the control variable.

a. Solution by conventional dynamic programming

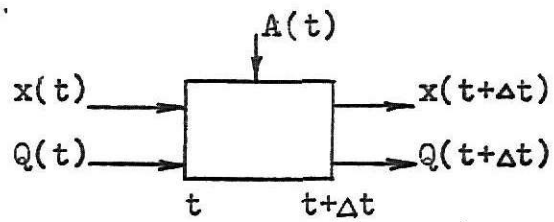
The range and step size for x(t), Q(t) and A(t) used are as follows:

	range	number of intervals	step size
x(t)	0-100	20	5.0
Q(t)	0-150	30	5.0
A(t)	0-6.0	30	0.2

The number of stages considered are 5 and 10. The step sizes in x and Q given above are suitable for both 5 and 10 stages. A 20-stage process was not considered because the computation time increases to more than 150 minutes.

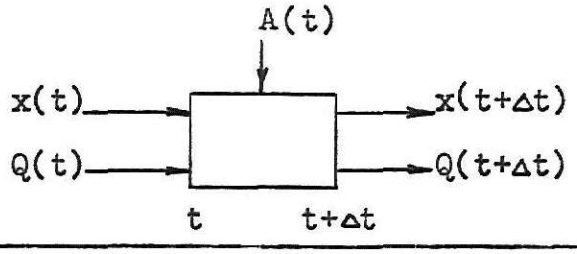
Solutions were obtained on an IBM 360 computer. The computer program used and a sample of the computer printout of the results for the 10-stage process is given in Appendix B. The flowchart for this program is similar to the one given in Fig. 19, Appendix A for the earlier model, except that additional logic for the second variable and a two-dimensional interpolation is added to it. The results for the given initial condition were interpolated manually and are given in Tables 9 and 10. The same results are plotted in Figs. 12 through 15. As can be seen from these graphs and tables, the results of the 10-stage process are a significant improvement over the results of the 5-stage process. The reason is that a 5-stage process with two state variables is a very approximate representation of the real situation. The profit increases from 500 to 618, an increase of 23.4%. Sales rate also increases from 110.7 to 121.5, an increase of 9.8%. The reason is that for a 10-stage process, the advertising expenditure is utilized in a better manner because the effect of advertising is felt much sooner than in the case of a 5-stage process.

Table 9. 5-stage process by conventional dynamic programming.



t - t+Δt	x(t)	Q(t)	A(t)	x(t+Δt)	Q(t+Δt)	P(t)	S_T
0 - .2	20.0	20.0	6.0	30.0	42.9	70	-21
.2 - .4	30.0	42.9	6.0	39.5	76.0	90	-26
.4 - .6	39.5	76.0	0	46.5	88.6	110	124
.6 - .8	46.5	88.6	0	54.3	100.3	130	301
.8 - 1.0	54.3	100.3	0	64.3	110.7	150	500

Table 10. 10-stage process by conventional dynamic programming.



t - t+Δt	x(t)	Q(t)	A(t)	x(t+Δt)	Q(t+Δt)	P(t)	S_T
0 - .1	20.0	20.0	6.0	25.0	32.5	70	-11
.1 - .2	25.0	32.5	6.0	29.8	50.3	80	-20
.2 - .3	29.8	50.3	6.0	34.0	70.1	90	-16
.3 - .4	34.0	70.1	5.2	37.1	90.2	100	- 3
.4 - .5	37.1	90.2	0	39.2	96.4	110	77
.5 - .6	39.2	96.4	0	41.1	102.5	120	167
.6 - .7	41.1	102.5	0	43.3	107.3	130	272
.7 - .8	43.3	107.3	0	46.1	112.1	140	385
.8 - .9	46.1	112.1	0	49.5	117.2	150	500
.9 - 1.0	49.5	117.2	0	54.0	121.5	160	618

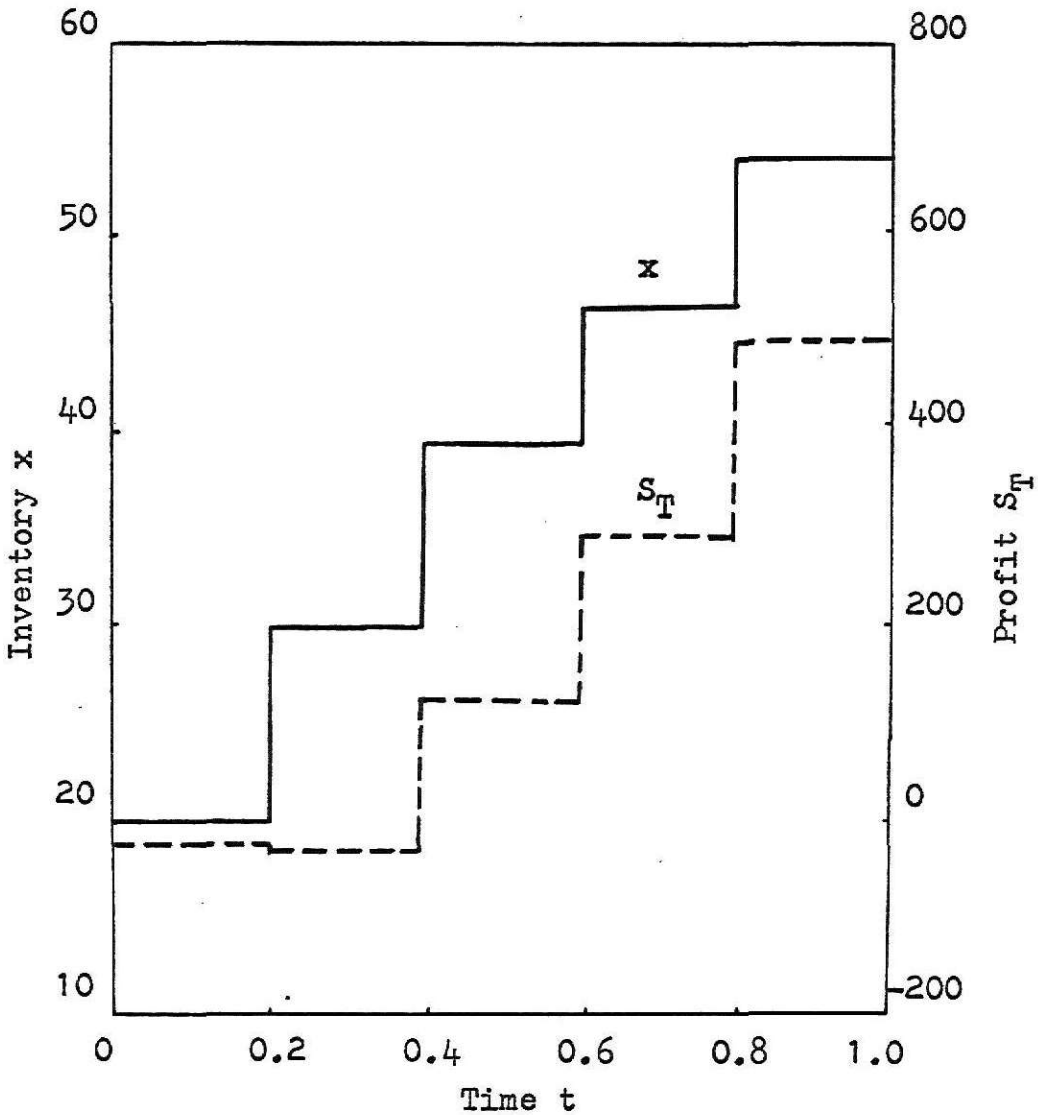


Fig. 12. Inventory and profit for 5-stage process by conventional dynamic programming

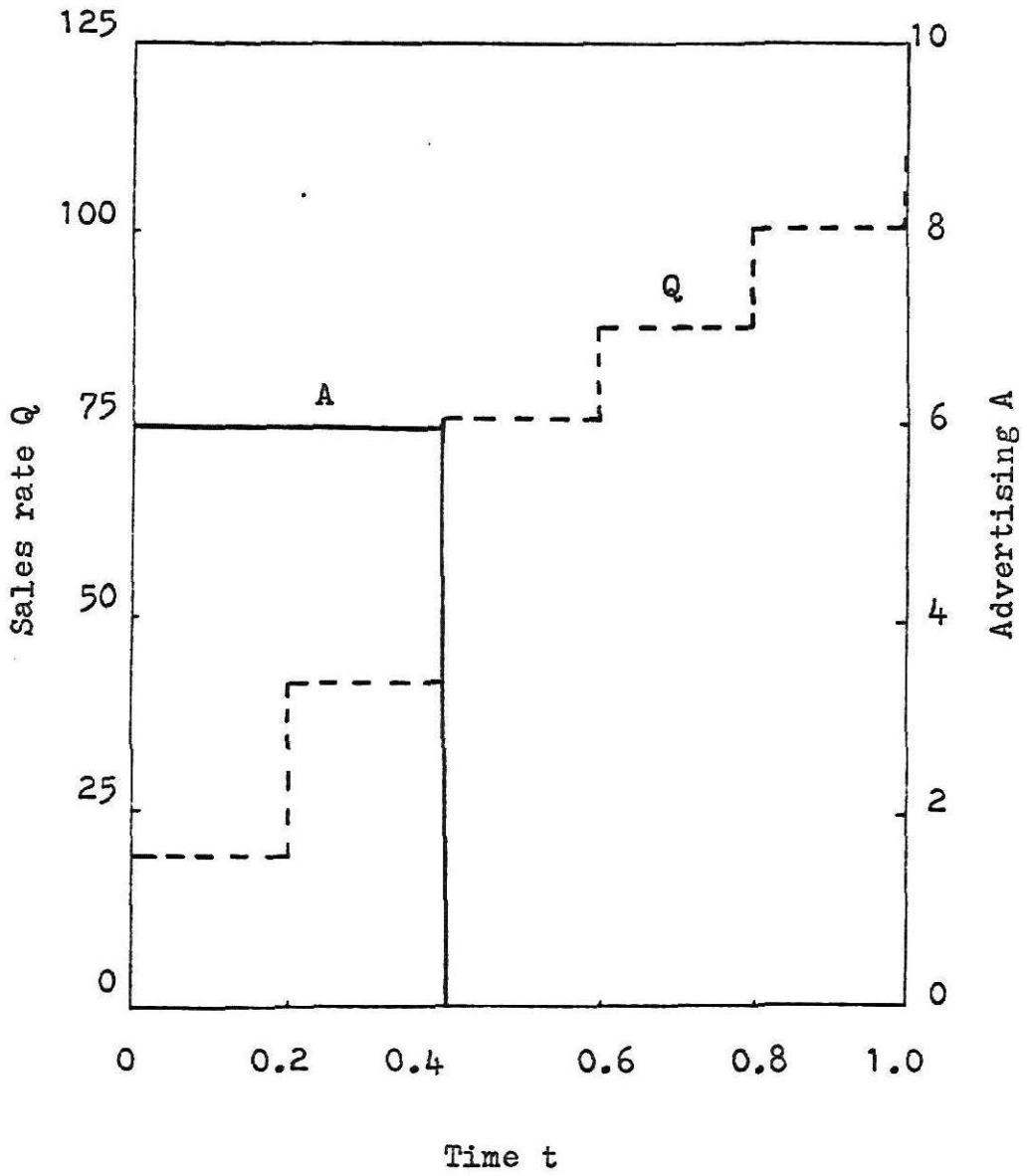


Fig. 13. Sales rate and advertising for 5-stage process by conventional dynamic programming

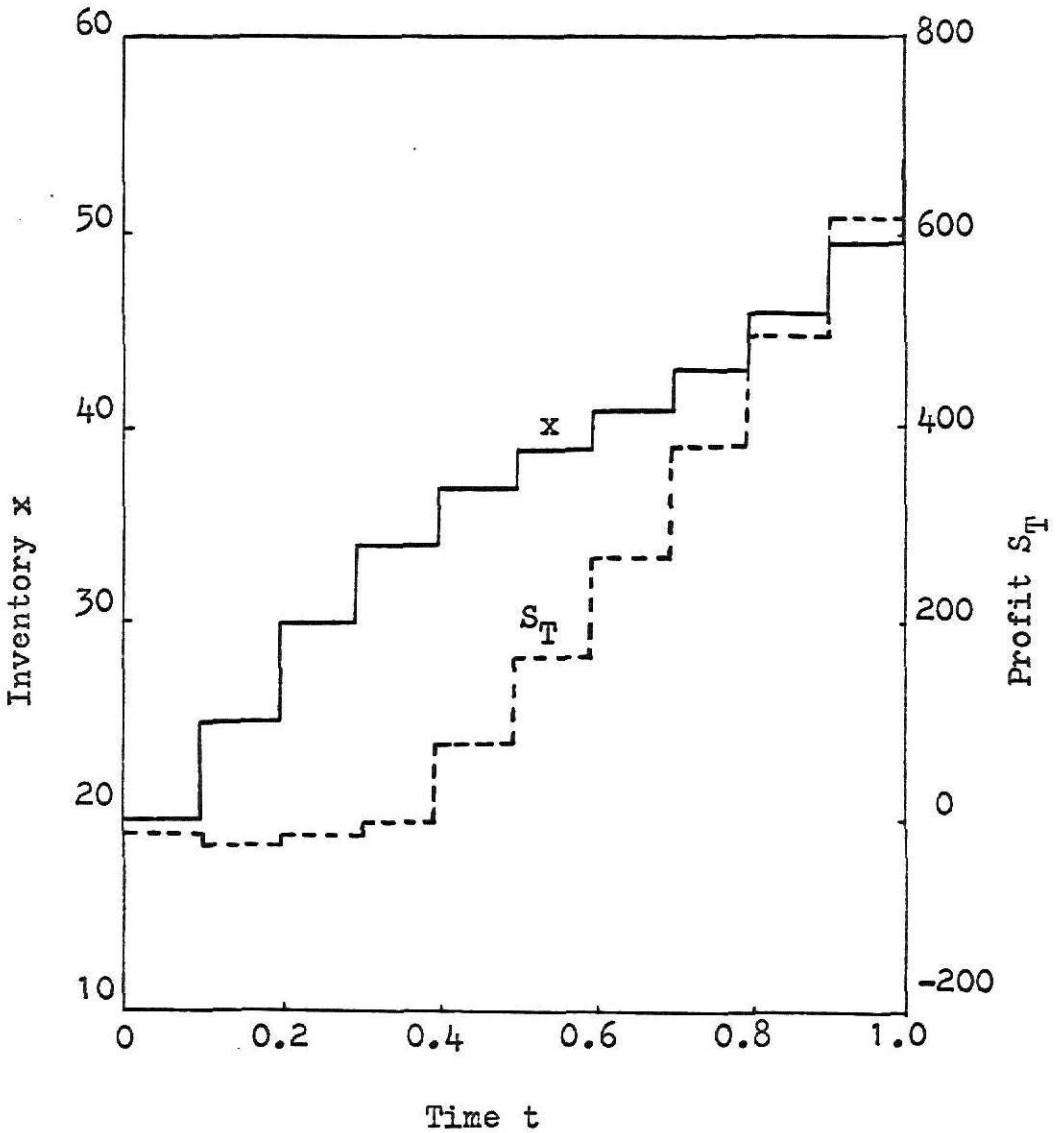


Fig. 14. Inventory and profit for 10-stage process by conventional dynamic programming.

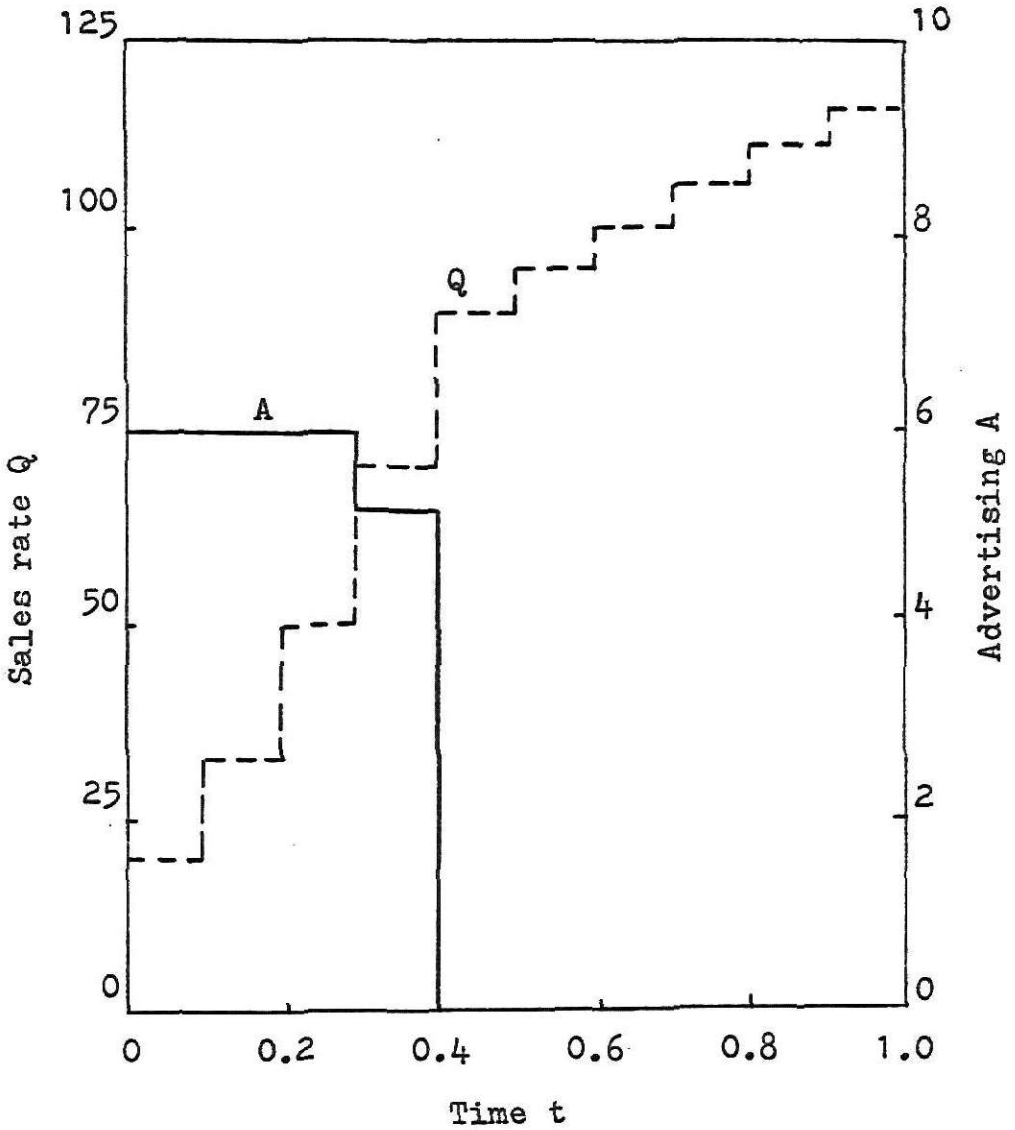


Fig.15. Sales rate and advertising for 10-stage process by conventional dynamic programming

b.Solution by state increment dynamic programming

A solution is next obtained by state increment dynamic programming. From the earlier solution by conventional dynamic programming, it is known that both the state variables x(t) and Q(t) increase with time. This, then, is the preferred direction of motion of x(t) and Q(t). This knowledge is used in formulating the computational scheme.

The size of one time block is taken to be $\Delta T \times \Delta x \times \Delta Q$, where $\Delta T = t_f - t_o = 1$. One such block for the 5-stage process is shown in fig. 16. The time interval ΔT is further divided into 5 smaller increments of Δt so that

$$5\Delta t = \Delta T = 1, \text{ or } t = 0.2$$

Because the preferred direction of motion is such that x(t) and Q(t) increase with time, we first process the blocks with the largest values of x and Q and then proceed towards the origin as follows. The first block to be processed has the largest values of x, Q and t. In this block, x and Q are first kept fixed and optimal function is found for all the 5 time increments (stages). The processing then proceeds to the next lower value of Q (keeping x still fixed) and processing another block as before. When all the blocks for all different values of Q have been processed, value of x is decreased to its next lower value and all the processing described above is again repeated. Processing is completed when all the values of x and Q have been processed for all values of t.

In processing a next block, the values of the optimal profit at the boundary points common to this block and the

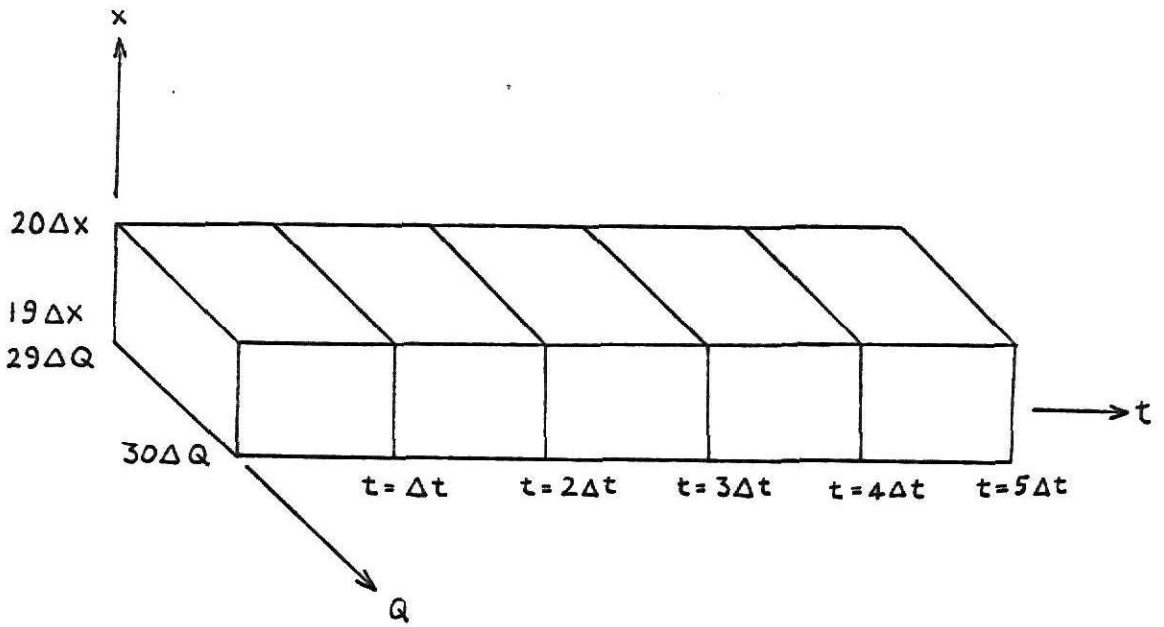


Fig. 16. Block for x-Q space.

previous adjacent block are used as starting values for the current block.

The procedure for obtaining the starting values for the first block is similar to the procedure described in the earlier model.

Figure 16a illustartes in a simplified manner, the order in which the blocks are processed.

The computer program in FORTRAN language and a sample prinout of the results for the 10-stage process are given in Appendix B. The flowchart for this program is similar to the one given in Fig. 20, Appendix A for the earlier model, except that additional logic for the second variable and a two-dimensional interpolation is added to it.

The results for the initial given condition were interpolated manually and are given in Tables 11 and 12. The results for the 5-stage process are also plotted in Figs. 17 and 18. Since the results of the 10-stage process by the new method are not significantly different from the results obtained by conventional dynamic programming, these are not separately plotted.

c. Comperison

For the 5-stage process, the results obtained by the new method are far inferior to those obtained by conventional dynamic programming. Optimal profit is 365 which is lower by 27% compared to the value of 500 obtained by conventional dynamic programming. The reason is that 5 stages are too small for the new method

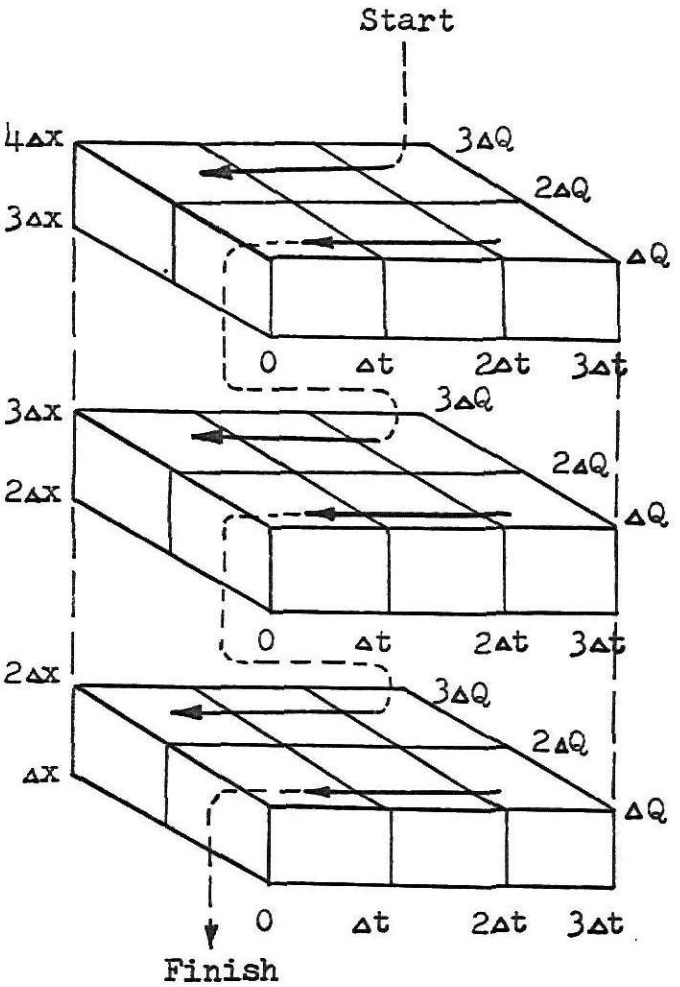
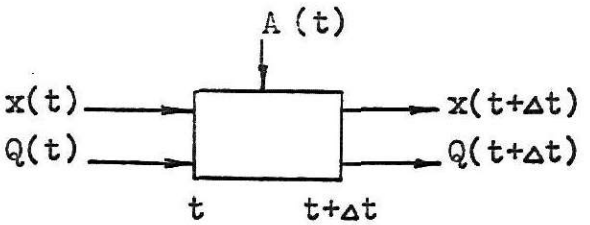


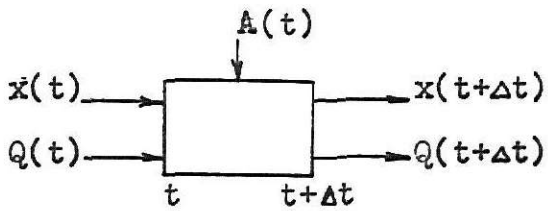
Fig. 16a. Order of processing the blocks.

Table 11. Results of 5-stage process using state increment dynamic programming.



t - t+Δt	x(t)	Q(t)	A(t)	x(t+Δt)	Q(t+Δt)	P(t)	S_T
0 - .2	20.0	20.0	6.0	30.9	42.9	70	-81
.2 - .4	30.0	42.9	6.0	39.5	76.0	90	-136
.4 - .6	39.5	76.0	0	46.5	88.6	110	-4
.6 - .8	46.5	88.6	0	54.3	100.3	130	166
.8 - 1.0	54.3	100.3	0	64.3	110.7	150	365

Table 12. Results of 10-stage process using state increment dynamic programming.



t - t+Δt	x(t)	Q(t)	A(t)	x(t+Δt)	Q(t+Δt)	P(t)	S_T
0 - .1	20.0	20.0	6.0	25.0	32.5	70	-19
.1 - .2	25.0	32.5	6.0	29.8	50.3	80	-26
.2 - .3	29.8	50.3	6.0	34.0	70.1	90	-21
.3 - .4	34.0	70.1	5.2	37.1	90.2	100	- 9
.4 - .5	37.1	90.2	0	39.2	96.4	110	73
.5 - .6	39.2	96.4	0	41.1	102.5	120	163
.6 - .7	41.1	102.5	0	43.3	107.3	130	268
.7 - .8	43.3	107.3	0	46.1	112.1	140	381
.8 - .9	46.1	112.1	0	49.5	117.2	150	496
.9 - 1.0	49.5	117.2	0	54.0	121.5	160	613

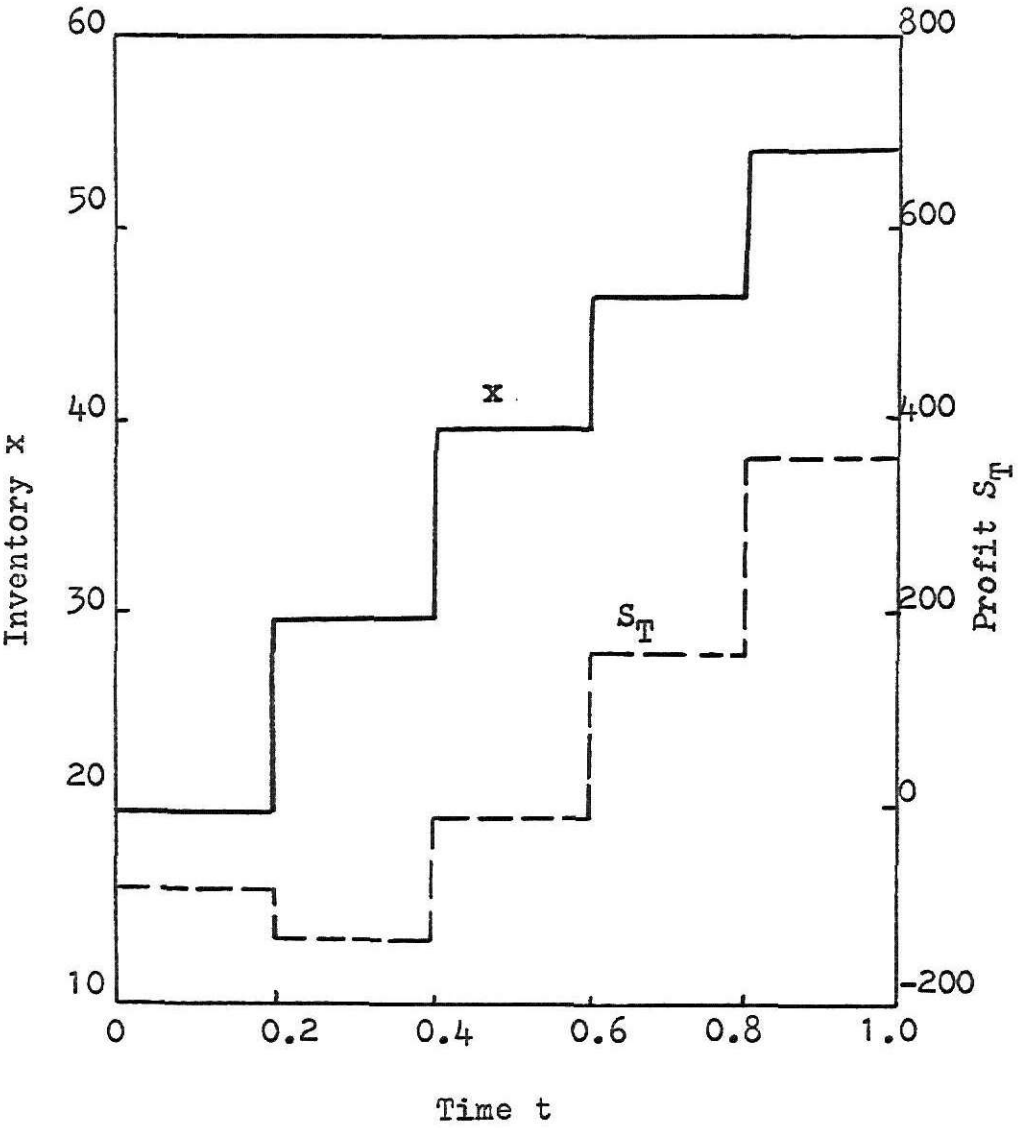


Fig. 17. Inventory and profit for 5-stage process and state increment dynamic programming.

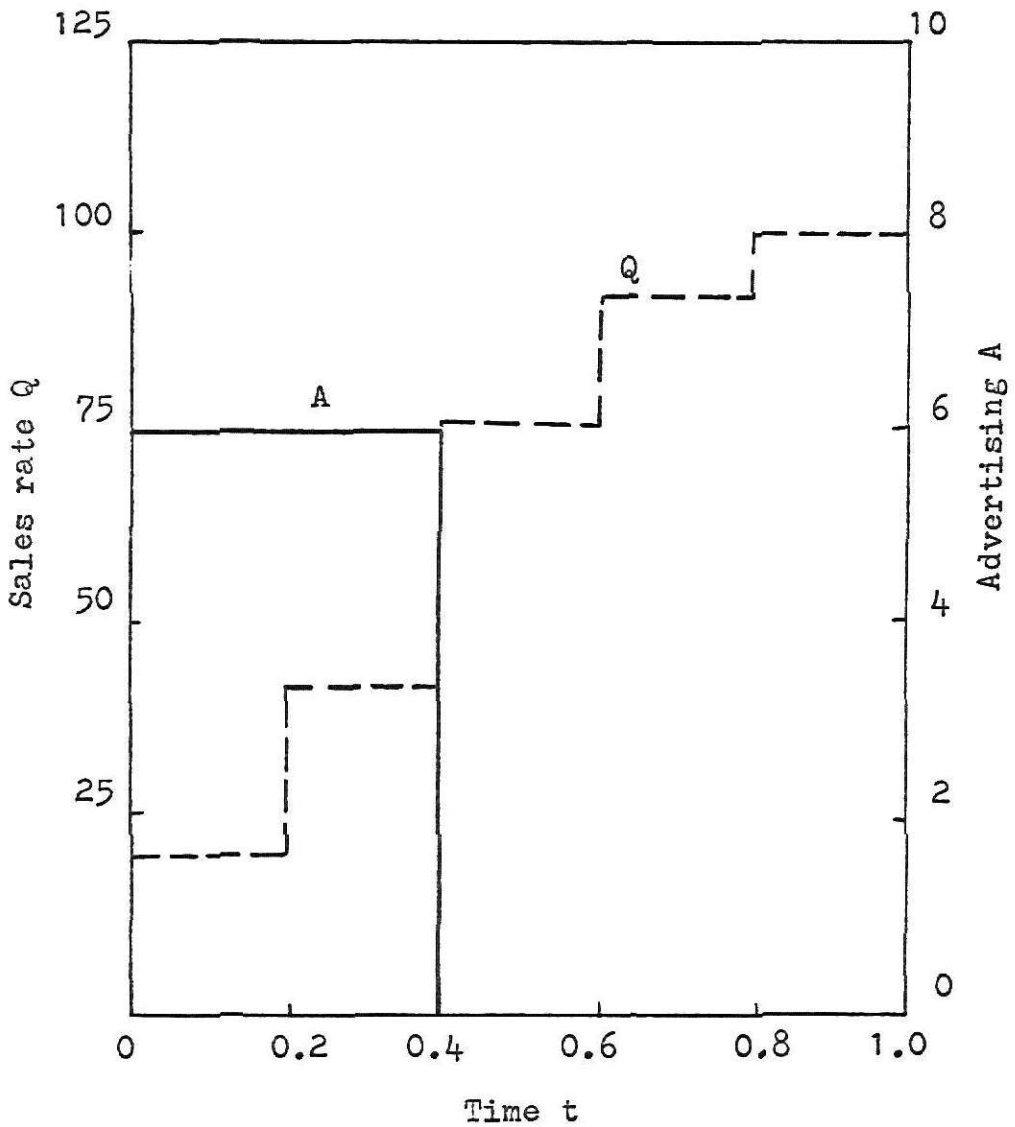


Fig. 18. Sales rate and advertising for 5-stage process and state increment dynamic programming.

which involves considerable interpolation.

The values of x and Q are same in both the methods, the reason being that in determining the next states of x and Q, the same difference equations are used.

However, when the number of stages is increased to 10, the results obtained by the new method are comparable to those obtained by conventional dynamic programming. The optimal profit is 613 which is lower by 1.1% compared to the value of 618 obtained by dynamiv programming. The values of x and Q are again same in both the methods. However the value of S_T is different because this value is computed by a different scheme in the new method.

The computation time taken by the new method is somewhat higher as compared to,the conventional dynamic programming. The increase is 10.8% for 5 stages and 5.6% for 10 stages. This increase is reasonable considering the greater complexity of computer programs for the new method.

The number of fast memory locations required in the state increment dynamic programming is considerably less than that required by conventional dynamic programming. In conventional dynamic programming, 1200 locations are needed whereas in the new method only 36 locations are needed for 5 stages and 66 for 10 stages. This is a considerable saving in the fast memory locations.

A summary of the comparison of the results by the two methods is given in Table 13.

Table 13. Comparison of the results.

Number of stages	Optimal Revenue S_T		Computation time in minutes		Number of fast storage locations used	
	DP	SI	DP	SI	DP	SI
5	500	365	9.27	10.27	1200	36
10	618	613	17.68	18.70	1200	66

DP: By dynamic programming

SI: By state increment programming

DISCUSSION

The accuracy of the results obtained by the state increment method is nearly the same as that of conventional dynamic programming when the increment size in the state variables, the set of admissible controls, the increment size in time (or the number of stages) and the accuracy of the interpolation formula for the optimal function are all the same. The major drawback of the state increment method is that the average value of the time interval over which control is applied must be about the same order of magnitude as the fixed time increment to get accurate results. This condition cannot be always fulfilled. The reason is that even if one of the state variables changes more rapidly than others, the next state always tends to increment along this particular state variable because this state requires the least time to change by one increment. When such a condition occurs, the state increment method leads to inaccurate resluts due to inaccuracies in interpolation.

Such a condition is illustrated in Fig. 18a. Suppose that the process is such that state x_1 increases rapidly in the earlier stages but slowly in the later stages. For state x_2 the reverse is the case. If the state increment dynamic programming is used for such a process, results will be inaccurate. First, it is difficult to determine a proper step size for both x_1 and x_2. A step size which is suitable for the earlier stages is clearly unsuitable for the later stages.Second, because x_1 changes so rapidly in the earlier stages, the next state will always lie along the x_1 axis. In other words, the value of Δx_1

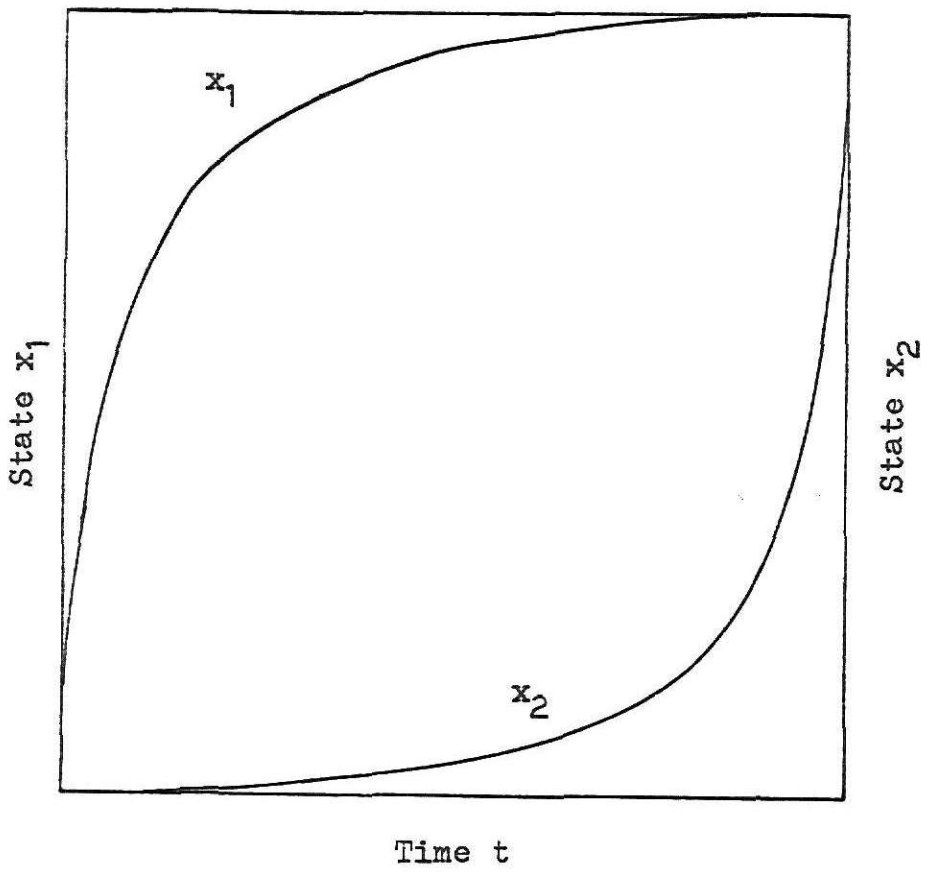


Fig. 18a. An example of a not well-behaved process.

is too small at this time. This condition is the same as that illustrated in Fig. 11a. Thus the results obtained will be mostly by extrapolation and hence inaccurate. During the later stages, similar situation exists because x_2 increases rapidly.

A word about the assumptions on which all the three models are built. It is true that under particular situations, it might be felt that some of the assumptions made in the models are inadequate and thus do not represent the actual conditions. However, it may be mentioned that the primary purpose of this report is not to develop these models but rather to demonstarte that the application of the state increment technique to such type of models is feasible and practical. This, it is felt, has been achieved. It is not concluded however, that the treatment by the state increment technique to the models presented in this report has been an exhausitive one. The state increment technique is a very recently developed technique with very little computational experience. In general, simplifications of the procedure occur in specific cases and many of these simplifications could be generalized to a significant class of problems. By this means, the effectiveness of the new technique can be still further increased. Another area of investigation is the general problem of the preferred direction of motion and for treating motion in the non-preferred direction. Thus this technique is still in its initial stages and improvements in its theory will provide in the future for a more powerful tool in a wider range of applications.

6. ACKNOWLEDGEMENT

The author wishes to express his deep sense of appreciation to his major professor, Dr. E.S. Lee for his guidance, constructive criticism, helpful suggestions and the personal interest taken in the preparation of this master's report.

7. REFERENCES

1. Aris, R., The Discrete Dynamic Programming, Blaisdell, New York, 1964.
2. Bellman, R., Dynamic Programming, Princeton University Press, New Jersey, 1957.
3. Bellman, R., and Kalaba, R., Dynamic Programming and Control Theory, Academic Press, New York, 1965.
4. Bellman, R., and Dreyfus, S.E., Applied Dynamic Programming, Princeton University Press, New Jersey, 1962.
5. Dreyfus, S.E., "Computational Aspects of Dynamic Programming", Operations Research, Vol. V, No. 3, June 1957, pp. 409-415.
6. Hildebrand, F.B., Introduction to Numerical Analysis, McGraw Hill, New York, 1956.
7. Howard, R.A., Dynamic Programming And Markoff Processes, John Wiley, New York, 1960.
8. Larson, R.E., "An Approach to Reduce the High Speed Memory Requirement of Dynamic Programming", Jour. of Math. Anal. & Appl. Sci., Vol. 2, 1965, pp. 519-37.
9. Larson, E.S., "Dynamic Programming with Continuous Independent Variable", Doctoral Dissertation, Stanford University, Stanford, California.
10. Lee, E.S., "Dynamic Programming and Polynomial Approximation", an unpublished paper, Philips Petroleum Co., Bartlesville, Oklahoma.

11. Lee, E.S., Quasilinearization and Invariant Imbedding, Academic Press, New York, 1968.

12. Lee, E.S., and Shaikh, M.A., "Optimal Production Planning by a Gradient Technique I. First Variations", an unpublished paper, Kansas State University, Manhattan, Kansas.

13. Lee, E.S., "Dynamic Programming", Lecture Notes, Kansas State University, Manhattan, Kansas, 1968.

14. Nemhauser, G.L., Introduction to Dynamic Programming, John Wiley, New York, 1966.

15. Teichroew, D., An Introduction to Management Science-Deterministic Models, John Wiley, New York, 1966.

16. Spradlin, B.C., and Pierce, D.A., "Production Scheduling under a Learning Effect by Dynamic Programming", Journal of Industrial Engineering, Vol. XVIII, No. 3, March, 1967, pp. 219-222.

APPENDIX A

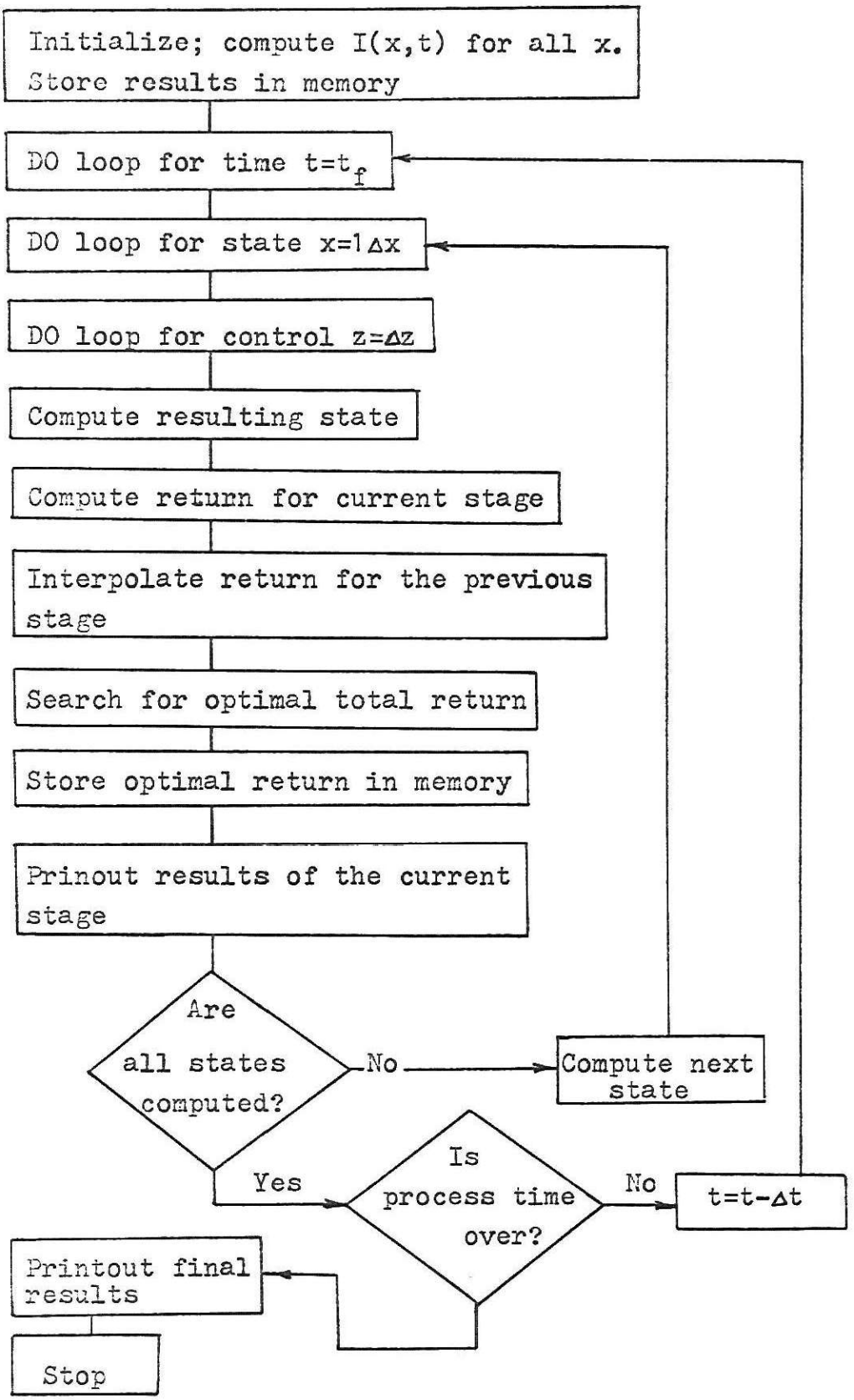


Fig. 19. Flowchart for conventional dynamic programming

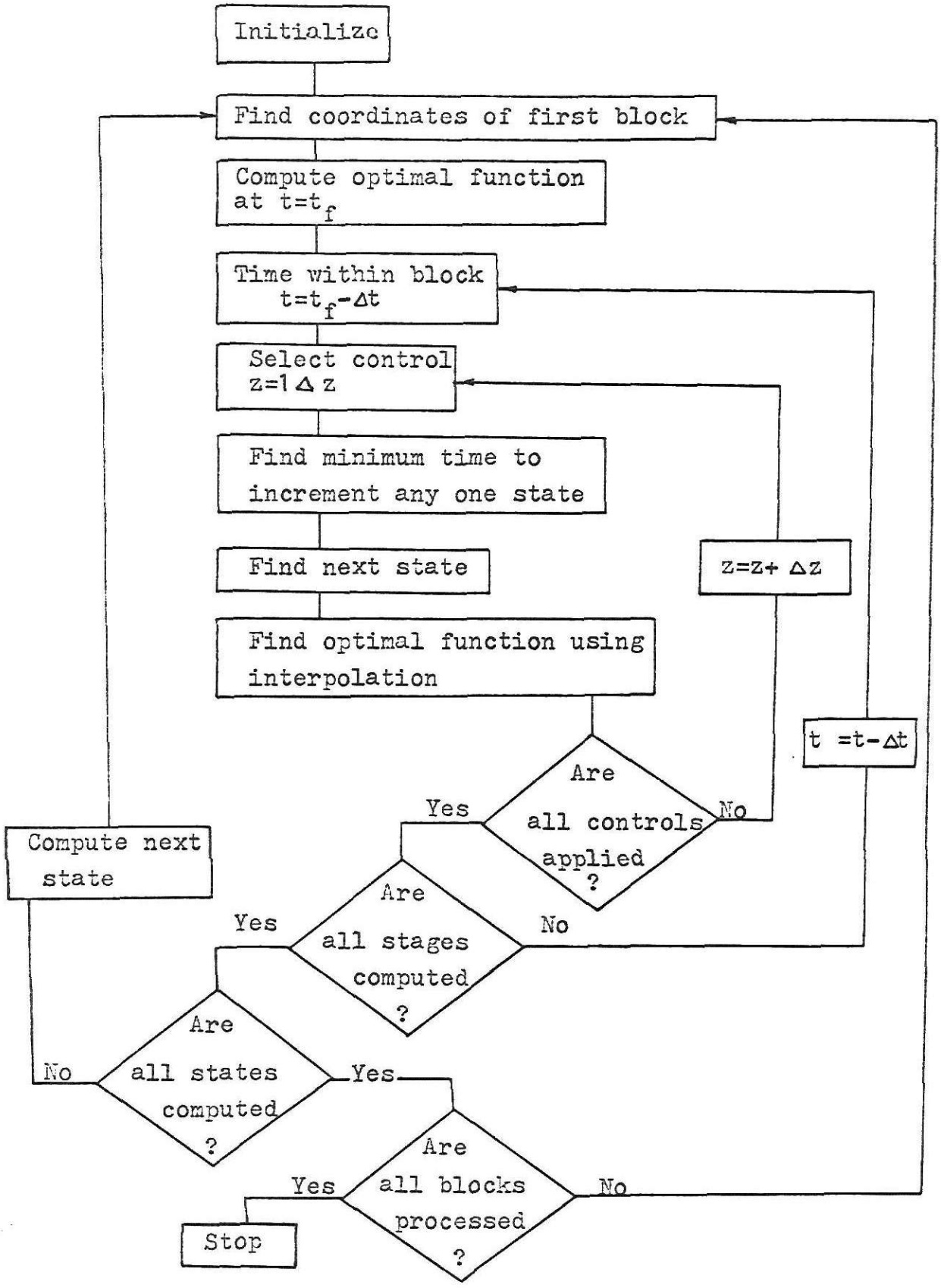


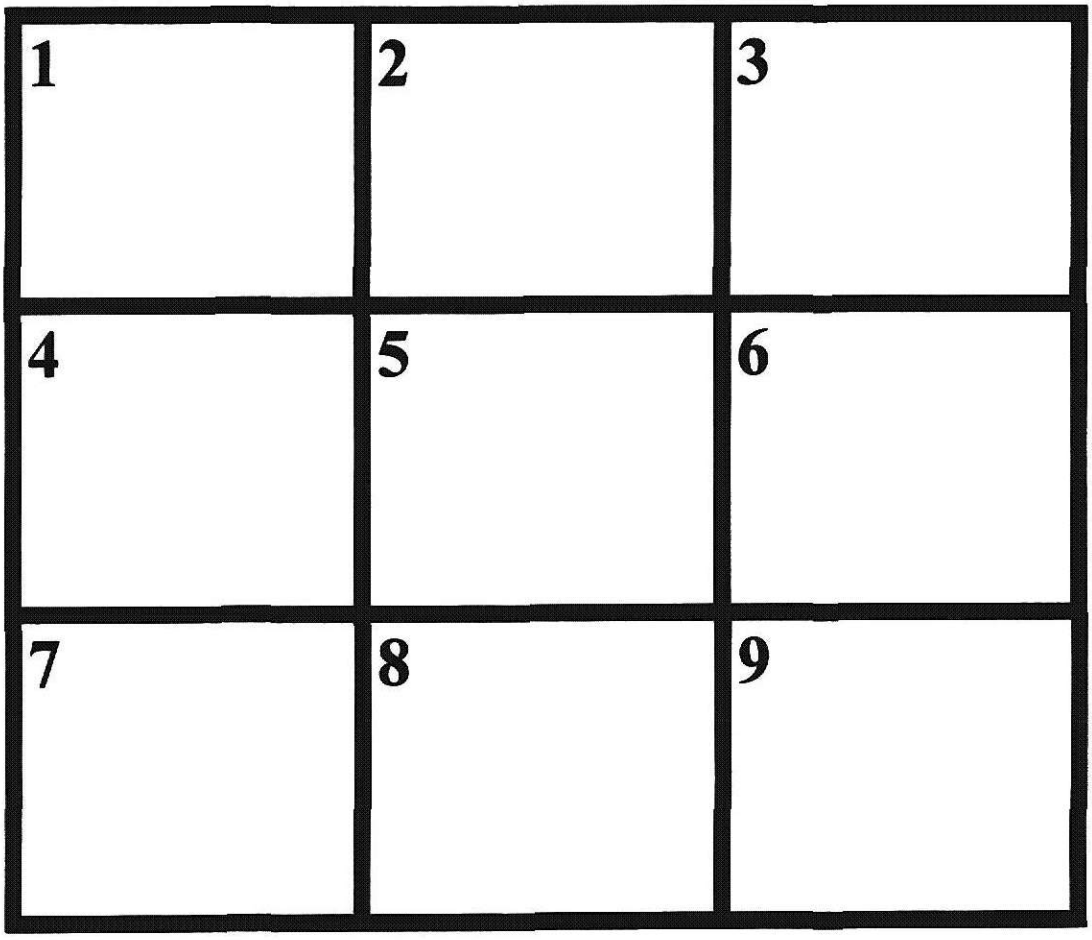
Fig. 20. Flowchart for state increment dynamic programming

APPENDIX B

OVERSIZED DOCUMENT

THE FOLLOWING DOCUMENTS ARE BEING FILMED IN SECTIONS.

THE FOLLOWING IMAGES WILL BE TAKEN FROM LEFT TO RIGHT, TOP TO BOTTOM. SEE EXAMPLE BELOW:



```
      $JOB            VARI,RUN=CHECK,TIME=10,PAGES=50
      C
      C     COMPUTER PROGRAM AND SAMPLE RESULTS FOR A PRODUCTION
      C     SCHEDULING MODEL UNDER LEARNING EFFECT
      C
 1          DIMENSION XLAST(100)
 2          XMAX=100.
 3          XMIN=10.
 4          MAXX=XMAX
 5          MINX=XMIN
 6          E=XMAX
 7          F=XMIN
 8          A=.234
 9          B=625.
10          C=1.
11          D=100.
12          DIMENSION S(100),XSIZE(100)
13          DIMENSION SP(100)
14          OLDC=99999999.
15          AI=1.
16          I=AI
17       10 FORMAT (10X,F10.1,10X,F10.2,10X,F10.1)
18       11 FORMAT('           OPTIMAL TABLE FOR PRODUCTION LOTS,N=',I2)
19       12 FORMAT('           PRODUCTION                COST                PRODUCTI
           ION')
20       13 FORMAT ('             IN N LOTS             IN N LOTS           IN (N-I
           1) LOTS')
21          DO 70 J=1,100
22       70 XLAST(J)=0.
23          DO 1 K=MINX,MAXX
24          X=K
25          BB=D/(.5**A)
26          BBB=D/(.5+X/2.)**A
27          BBBB=D/(.5+X)**A
28          COST=B+C*X**2+(X/6.)*(BB+4.*BBB+BBBB)
29          S(K)=COST
30          XSIZE(K)=X
31        1 CONTINUE
32          DO 4 K=MINX,MAXX
33        4 SP(K)=S(K)
34          PRINT 11,I
35          PRINT 12
36          PRINT 13
37          PRINT 10, (XSIZE(M),S(M),XLAST(M),M=MINX,MAXX)
38        7 AI=AI+1.
39          I=AI
40          MM=AI*F
41          DO 2 K=MM,MAXX
42          FK=K
43          KA=K-F*(I-1)
44          COUNNY=99999999.
45          DO 3 KB=MINX,KA
46          X=KB
47          AA=FK-X-F*(AI-1.)+.5
48          H=X/2.
49          BB=D/(AA**A)
50          BBB=D/(AA+H)**A
51          BBBB=D/(AA+2.*H)**A
52          COST=B+C*X**2+(H/3.)*(BB+4.*BBB+BBBB)
53          KK=K-KB
```

```
AGES=50
TS FOR A PRODUCTION
FFECT

10.1)
OR PRODUCTION LOTS,N=',I2)
       COST                PRODUCTI

     IN N LOTS              IN (N-1

BB)

M=MINX,MAXX)

BB)
```

```
54          COST1=SP(KK)
55          COST=COST+COST1
56          IF(COST.GT.COUMMY) GO TO 3
57          COUMMY=COST
58          XX=X
59          XX=KK
60        3 CONTINUE
61          S(K)=COUMMY
62          XLAST(K)=XX
63        2 XSIZE(K)=FK
64          PRINT 11,I
65          PRINT 12
66          PRINT 13
67          PRINT 10,(XSIZE(K),S(K),XLAST(K),K=MM,MAXX)
68          G=S(100)
69          GP=SP(100)
70          DO 5 KK=MINX,MAXX
71        5 SP(KK)=S(KK)
72          IF(G.LT.GP) GO TO 7
73          STOP
74          END
```

OPTIMAL TABLE FOR PRODUCTION LOTS,N= 1

PRODUCTION IN N LOTS	COST IN N LOTS	PRODUCTION IN (N-1) LOTS
10.0	1464.52	0.0
11.0	1547.32	0.0
12.0	1631.23	0.0
13.0	1716.33	0.0
14.0	1802.69	0.0
15.0	1890.39	0.0
16.0	1979.47	0.0
17.0	2069.99	0.0
18.0	2161.98	0.0
19.0	2255.49	0.0
20.0	2350.54	0.0
21.0	2447.16	0.0
22.0	2545.37	0.0
23.0	2645.21	0.0
24.0	2746.69	0.0
25.0	2849.82	0.0
26.0	2954.63	0.0
27.0	3061.13	0.0
28.0	3169.34	0.0
29.0	3279.27	0.0
30.0	3390.92	0.0
31.0	3504.32	0.0
32.0	3619.46	0.0
33.0	3736.37	0.0
34.0	3855.05	0.0
35.0	3975.51	0.0
36.0	4097.75	0.0
37.0	4221.79	0.0
38.0	4347.63	0.0
39.0	4475.27	0.0
40.0	4604.73	0.0
41.0	4736.00	0.0
42.0	4869.11	0.0
43.0	5004.04	0.0
44.0	5140.80	0.0
45.0	5279.41	0.0
46.0	5419.86	0.0
47.0	5562.15	0.0
48.0	5706.30	0.0
49.0	5852.30	0.0
50.0	6000.16	0.0
51.0	6149.89	0.0
52.0	6301.48	0.0
53.0	6454.94	0.0
54.0	6610.27	0.0
55.0	6767.48	0.0
56.0	6926.57	0.0
57.0	7087.53	0.0
58.0	7250.38	0.0
59.0	7415.12	0.0
60.0	7581.74	0.0
61.0	7750.25	0.0
62.0	7920.66	0.0
63.0	8092.96	0.0
64.0	8267.16	0.0
65.0	8443.26	0.0

```
$JOB            VARI,RUN=CHECK,TIME=9,PAGES=100,LINES=60
C
C       COMPUTER PROGRAM AND SAMPLE RESULTS FOR AN INVENTORY
C       MODEL WITH ONE STATE VARIABLE BY CONVENTIONAL
C       DYNAMIC PROGRAMMING
C
        T=1.
        A=2.
        B=1.
        CI=0.1
        C=5.
        XM=10.
        CP=0.001
        PM=5.
        DELT=.1
        DELX=.5
        DELZ=0.1
        OLDS=99999999.
        NT=10
        NZ=101
        NX=21
        DIMENSION S(101),ZBEST(101),X(101),XP(101),SP(101),XEXIT(101)
        DIMENSION QQ(101)
  100 FORMAT (5X,3(F6.2,3X),F6.3,3X,F6.2)
  101 FORMAT ('       OPTIMAL TABLE AT TIME =',F4.2)
  102 FORMAT('       X(N+1)     Z(N)      X(N)      COST     SALES')
        DO 15 L=1,101
   15 SP(L)=0.
        DO 10 J=1,NT
        DT=J
        DT=DT*DELT
        Q=A+B*(T-DT+DELT/2)
        M=0
        DO 11 N=1,NX
        XIN=N-1
        XIN=XIN*DELX
        DO 12 K=1,NZ
        Z=K-1
        Z=Z*DELZ
        XOUT=XIN+DELT*(Z-Q)
        IF(XOUT.GT.10.) GO TO 12
        IF(XOUT.LT.0.0) GO TO 12
        S1=(CI*(XM-XOUT/2.-XIN/2.)**2+CP*EXP((PM-Z)**2))*DELT
        L=XOUT/DELX
        AL=L
        S2=SP(L+1)+(SP(L+2)-SP(L+1))*(XOUT/DELX-AL)
        SNEW=S1+S2
        IF(SNEW.GT.OLDS) GO TO 12
        OLDS=SNEW
        ZZ=Z
        XX=XOUT
   12 CONTINUE
        M=M+1
        S(M)=OLDS
        ZBEST(M)=ZZ
        X(M)=XIN
        XEXIT(M)=XX
        QQ(M)=Q
   11 OLDS=99999999.
        DO 23 N=1,NX
```

```
55          XP(N)=X(N)
56       23 SP(N)=S(N)
57          TIME=T+DELT-DT
58          PRINT 101,TIME
59          PRINT 102
60          DO 50 M=1,NX
61       50 PRINT 100,X(M),ZBEST(M),XEXIT(M),S(M),QQ(M)
62       10 CONTINUE
63          STOP
64          END
```

OPTIMAL TABLE AT TIME =1.00

X(N+1)	Z(N)	X(N)	COST	SALES
0.00	6.80	0.39	0.964	2.95
0.50	6.80	0.89	0.869	2.95
1.00	6.80	1.38	0.778	2.95
1.50	6.80	1.88	0.693	2.95
2.00	6.80	2.38	0.612	2.95
2.50	6.70	2.88	0.537	2.95
3.00	6.70	3.38	0.466	2.95
3.50	6.70	3.88	0.400	2.95
4.00	6.70	4.38	0.340	2.95
4.50	6.70	4.88	0.284	2.95
5.00	6.60	5.36	0.233	2.95
5.50	6.60	5.86	0.188	2.95
6.00	6.60	6.36	0.147	2.95
6.50	6.50	6.85	0.111	2.95
7.00	6.50	7.35	0.081	2.95
7.50	6.40	7.84	0.055	2.95
8.00	6.40	8.34	0.034	2.95
8.50	6.30	8.84	0.018	2.95
9.00	6.10	9.31	0.007	2.95
9.50	5.90	9.80	0.001	2.95
10.00	2.90	9.99	0.008	2.95

OPTIMAL TABLE AT TIME =0.90

X(N+1)	Z(N)	X(N)	COST	SALES
0.00	7.10	0.43	1.849	2.85
0.50	7.00	0.92	1.663	2.85
1.00	7.00	1.41	1.486	2.85
1.50	7.00	1.91	1.319	2.85
2.00	7.00	2.41	1.162	2.85
2.50	7.00	2.91	1.015	2.85
3.00	7.00	3.41	0.878	2.85
3.50	7.00	3.91	0.751	2.85
4.00	6.90	4.40	0.634	2.85
4.50	6.90	4.90	0.527	2.85
5.00	6.90	5.40	0.430	2.85
5.50	6.90	5.90	0.343	2.85
6.00	6.80	6.39	0.266	2.85
6.50	6.80	6.89	0.199	2.85
7.00	6.80	7.39	0.141	2.85
7.50	6.70	7.88	0.094	2.85
8.00	6.60	8.38	0.056	2.85
8.50	6.60	8.88	0.029	2.85
9.00	6.40	9.35	0.011	2.85
9.50	3.80	9.59	0.005	2.85
10.00	2.80	9.99	0.021	2.85

OPTIMAL TABLE AT TIME =0.80

X(N+1)	Z(N)	X(N)	COST	SALES
0.00	7.20	0.45	2.652	2.75
0.50	7.20	0.95	2.379	2.75
1.00	7.10	1.43	2.120	2.75
1.50	7.10	1.93	1.877	2.75
2.00	7.10	2.43	1.648	2.75
2.50	7.10	2.93	1.435	2.75
3.00	7.10	3.43	1.236	2.75
3.50	7.10	3.93	1.053	2.75
4.00	7.00	4.43	0.884	2.75
4.50	7.00	4.93	0.730	2.75
5.00	7.00	5.43	0.591	2.75

```
$JOB          VARI,RUN=CHECK,TIME=9,PAGES=100
C
C     COMPUTER PROGRAM AND SAMPLE RESULTS FOR AN INVENTORY
C     MODEL WITH ONE STATE VARIABLE BY STATE INCREMENT
C     DYNAMIC PROGRAMMING
C
      T=1.
      A=2
      B=1
      CI=.1
      C=5
      XM=10
      CP=.001
      PM=5
      DT=.1
      DX=.5
      DZ=.1
      OLDS=999
      NT=10
      NZ=101
      NX=31
      NDZ=DZ
      NS=NT
      XMAX=15
      ZMAX=10
      NBLOCK=0
      LINENO=0
      DIMENSION SB(12),SP(12)
  100 FORMAT ('1     TIME    X(N+1)    Z(N)      X(N)      COST    SALES')
  101 FORMAT (5X,4(F6.2,3X),F6.3,3X,F6.2)
  102 FORMAT ('      BLOCK NO.=',I3)
      PRINT 100
      DO 45 K=1,NT
   45 SB(K)=0
      SB(NT+1)=0
      DO 10 KX=1,NX
      NBLOCK=NBLOCK+1
      PRINT 102, NBLOCK
      XIN=NX-KX
      XIN=XIN*DX
      T=1.
      NS=NT
      Q=A+B*(T-.5*DT)
      DO 12 KZ=1,NZ
      Z=KZ*DZ
      XOUT=XIN+(Z-Q)*DT
      IF (XOUT.LT..001) GO TO 12
      SNEW=(CI*(XM-XOUT/2-XIN/2)**2+CP*EXP((PM-Z)**2))*DT
      IF (SNEW.GT.OLDS) GO TO 12
      OLDS=SNEW
      ZZ=Z
      XX=XOUT
   12 CONTINUE
      PRINT 101,T,XIN,ZZ,XX,OLDS,Q
      LINENO=LINENO+1
      IF (LINENO.LT.50) GO TO 42
      LINENO=0
      PRINT 100
   42 SP(NS)=OLDS
      OLDS=999
```

```
ES=100
S FOR AN INVENTORY
TATE INCREMENT

(N)      X(N)      COST      SALES')
)

EXP((PM-Z)**2))*DT
```

```
      NS=NS-1
      T=NS*DT
      Q=A+B*(T-.5*DT)
      DO 22 KZ=1,NZ
      Z=KZ*DZ
      H=Z-Q
      IF (H.EQ..0) GO TO 22
      DELT=ABS(DX/(Z-Q))
   51 XOUT=XIN+(Z-Q)*DELT
      XO=XIN+(Z-Q)*DT
      S1=(CI*(XM-XOUT/2-XIN/2)**2+CP*EXP((PM-Z)**2))*DELT
      S2=SB(NS+1)+(SB(NS+1)-SB(NS+2))*(DT-DELT)/DT
      IF(S2.LT..0) S2=0
      SNEW=S1+S2
      IF (SNEW.GT.OLDS) GO TO 22
      OLDS=SNEW
      ZZ=Z
      XX=XO
   22 CONTINUE
      PRINT 101,T,XIN,ZZ,XX,OLDS,Q
      LINENO=LINENO+1
      IF (LINENO.LT.50) GO TO 80
      LINENO=0
      PRINT 100
   80 CONTINUE
      IF (NS.LE.1) GO TO 43
      GO TO 42
   43 SP(NS)=OLDS
      DO 44 K=1,NT
   44 SB(K)=SP(K)
      OLDS=999
   10 CONTINUE
      STOP
      END
```

TIME	X(N+1)	Z(N)	X(N)	COST	SALES
BLOCK NO.= 1					
1.00	15.00	3.40	15.05	0.254	2.95
0.90	15.00	7.20	15.43	0.331	2.85
0.80	15.00	7.20	15.44	0.324	2.75
0.70	15.00	7.20	15.45	0.317	2.65
0.60	15.00	7.20	15.47	0.310	2.55
0.50	15.00	7.20	15.47	0.303	2.45
0.40	15.00	7.20	15.48	0.297	2.35
0.30	15.00	7.20	15.49	0.291	2.25
0.20	15.00	7.20	15.51	0.285	2.15
0.10	15.00	7.20	15.52	0.280	2.05
BLOCK NO.= 2					
1.00	14.50	3.40	14.55	0.206	2.95
0.90	14.50	5.40	14.76	0.453	2.85
0.80	14.50	7.10	14.93	0.589	2.75
0.70	14.50	7.20	14.95	0.586	2.65
0.60	14.50	7.20	14.97	0.574	2.55
0.50	14.50	7.20	14.97	0.561	2.45
0.40	14.50	7.20	14.98	0.549	2.35
0.30	14.50	7.20	14.99	0.538	2.25
0.20	14.50	7.20	15.01	0.527	2.15
0.10	14.50	7.20	15.02	0.517	2.05
BLOCK NO.= 3					
1.00	14.00	3.40	14.05	0.163	2.95
0.90	14.00	5.40	14.26	0.362	2.85
0.80	14.00	4.50	14.17	0.516	2.75
0.70	14.00	6.80	14.41	0.781	2.65
0.60	14.00	7.10	14.45	0.794	2.55
0.50	14.00	7.10	14.47	0.778	2.45
0.40	14.00	7.10	14.47	0.761	2.35
0.30	14.00	7.10	14.48	0.744	2.25
0.20	14.00	7.10	14.49	0.729	2.15
0.10	14.00	7.10	14.51	0.714	2.05
BLOCK NO.= 4					
1.00	13.50	3.40	13.55	0.125	2.95
0.90	13.50	5.40	13.76	0.282	2.85
0.80	13.50	4.50	13.67	0.402	2.75
0.70	13.50	3.80	13.61	0.614	2.65
0.60	13.50	3.80	13.63	0.564	2.55
0.50	13.50	7.10	13.97	0.953	2.45
0.40	13.50	7.10	13.97	0.935	2.35
0.30	13.50	7.10	13.98	0.915	2.25
0.20	13.50	7.10	13.99	0.895	2.15
0.10	13.50	7.10	14.01	0.876	2.05
BLOCK NO.= 5					
1.00	13.00	3.50	13.05	0.093	2.95
0.90	13.00	5.40	13.26	0.212	2.85
0.80	13.00	4.60	13.18	0.301	2.75
0.70	13.00	3.80	13.11	0.462	2.65
0.60	13.00	3.80	13.13	0.424	2.55
0.50	13.00	7.10	13.47	0.690	2.45
0.40	13.00	3.80	13.14	0.366	2.35
0.30	13.00	7.00	13.47	1.053	2.25
0.20	13.00	7.00	13.48	1.030	2.15
0.10	13.00	7.00	13.49	1.007	2.05

```
$JOB            VARI,RUN=CHECK,TIME=30,PAGES=150
C
C       COMPUTER PROGRAM AND SAMPLE RESULTS FOR AN INVENTORY AND
C       ADVERTISING MODEL WITH TWO STATE VARIABLES BY CONVENTIONAL
C       DYNAMIC PROGRAMMING
C
 1          OLDS=-989898.
 2          PI=50
 3          AN=150.2
 4          AB=70
 5          B=100
 6          C=2
 7          F=10
 8          CI=.15
 9          CA=1.5
10          XMAX=100
11          DT=.1
12          DX=5
13          DQ=5
14          DA=.2
15          NT=10
16          NX=21
17          NA=31
18          NQ=30
19          NQQ=NQ+1
20          NXX=NX+1
21      100 FORMAT (1H 12F10.2)
22      101 FORMAT (6H TIME=F6.2)
23      102 FORMAT ('       X(N)      Q(N)       X(N+1)     Q(N+1)      P(N)
           1A(N)    PROFIT')
24          DIMENSION SS(52,52),SP(52,52)
25          DO 10 L=1,NXX
26          DO 10 LL=1,NQQ
27       10 SP(L,LL)=0.
28          DO 16 J=1,NT
29          T=NT-J
30          T=T*DT
31          P=AB+B*T
32          PRINT 101,T
33          PRINT 102
34          DO 15 K=1,NX
35          X=K-1
36          X=X*DX
37          DO 20 L=1,NQ
38          Q=L-1
39          Q=Q*DQ
40          DO 25 N=1,NA
41          A=N-1
42          A=A*DA
43          QO=(Q*(1.+(C+A)*DT))/(1.+Q*(C+A)*DT/AN)
44          XO=X+(P-Q)*DT
45          IF (XO.LT..001) GO TO 25
46          S1=(Q*F-(PI-X)**2*CI-CA*A*Q)*DT
47          LX=XO/DX
48          IF (LX.GE.NX) GO TO 50
49          AX=LX
50          GO TO 51
51       50 AX=LX
52          LX=XMAX/DX
53       51 LQ=QO/DQ
```

```
      AQ=LQ
      B1=SP(LX+1,LQ+1)
      B2=SP(LX+2,LQ+1)
      B3=SP(LX+1,LQ+2)
      B4=SP(LX+2,LQ+2)
      B5=B1+(B2-B1)*(XQ/DX-AX)
      B6=B3+(B4-B3)*(XQ/DX-AX)
      S2=B5+(B6-B5)*(QQ/DQ-AQ)
      SNEW=S1+S2
      IF(SNEW.LT.OLDS) GO TO 25
      OLDS=SNEW
      AA=A
      XX=XQ
      QQ=QQ
   25 CONTINUE
      SS(K,L)=OLDS
      PRINT 100,X,Q,XX,QQ,P,AA,OLDS
   20 OLDS=-989898.
   15 CONTINUE
      DO 24 L=1,NX
      DO 24 LL=1,NQ
   24 SP(L,LL)=SS(L,LL)
      DO 33 L=1,NQ
   33 SP(NXX,L)=SP(NX,L)
      DO 34 L=1,NX
   34 SP(L,NQQ)=SP(L,NQ)
      SP(NXX,NQQ)=SP(NX,NQ)
   16 CONTINUE
      STOP
      END
```

E= 0.90

X(N)	Q(N)	X(N+1)	Q(N+1)	P(N)	A(N)	PROFIT
0.00	0.00	16.00	0.00	160.00	6.00	-37.50
0.00	5.00	15.50	5.96	160.00	0.00	-32.50
0.00	10.00	15.00	11.84	160.00	0.00	-27.50
0.00	15.00	14.50	17.65	160.00	0.00	-22.50
0.00	20.00	14.00	23.38	160.00	0.00	-17.50
0.00	25.00	13.50	29.03	160.00	0.00	-12.50
0.00	30.00	13.00	34.62	160.00	0.00	-7.50
0.00	35.00	12.50	40.13	160.00	0.00	-2.50
0.00	40.00	12.00	45.57	160.00	0.00	2.50
0.00	45.00	11.50	50.95	160.00	0.00	7.50
0.00	50.00	11.00	56.25	160.00	0.00	12.50
0.00	55.00	10.50	61.50	160.00	0.00	17.50
0.00	60.00	10.00	66.67	160.00	0.00	22.50
0.00	65.00	9.50	71.79	160.00	0.00	27.50
0.00	70.00	9.00	76.84	160.00	0.00	32.50
0.00	75.00	8.50	81.83	160.00	0.00	37.50
0.00	80.00	8.00	86.76	160.00	0.00	42.50
0.00	85.00	7.50	91.63	160.00	0.00	47.50
0.00	90.00	7.00	96.44	160.00	0.00	52.50
0.00	95.00	6.50	101.20	160.00	0.00	57.50
0.00	100.00	6.00	105.90	160.00	0.00	62.50
0.00	105.00	5.50	110.54	160.00	0.00	67.50
0.00	110.00	5.00	115.14	160.00	0.00	72.50
0.00	115.00	4.50	119.67	160.00	0.00	77.50
0.00	120.00	4.00	124.16	160.00	0.00	82.50
0.00	125.00	3.50	128.60	160.00	0.00	87.50
0.00	130.00	3.00	132.98	160.00	0.00	92.50
0.00	135.00	2.50	137.32	160.00	0.00	97.50
0.00	140.00	2.00	141.60	160.00	0.00	102.50
0.00	145.00	1.50	145.84	160.00	0.00	107.50
5.00	0.00	21.00	0.00	160.00	6.00	-30.37
5.00	5.00	20.50	5.96	160.00	0.00	-25.37
5.00	10.00	20.00	11.84	160.00	0.00	-20.37
5.00	15.00	19.50	17.65	160.00	0.00	-15.37
5.00	20.00	19.00	23.38	160.00	0.00	-10.37
5.00	25.00	18.50	29.03	160.00	0.00	-5.37
5.00	30.00	18.00	34.62	160.00	0.00	-0.37
5.00	35.00	17.50	40.13	160.00	0.00	4.63
5.00	40.00	17.00	45.57	160.00	0.00	9.63
5.00	45.00	16.50	50.95	160.00	0.00	14.63
5.00	50.00	16.00	56.25	160.00	0.00	19.63
5.00	55.00	15.50	61.50	160.00	0.00	24.63
5.00	60.00	15.00	66.67	160.00	0.00	29.63
5.00	65.00	14.50	71.79	160.00	0.00	34.63
5.00	70.00	14.00	76.84	160.00	0.00	39.63
5.00	75.00	13.50	81.83	160.00	0.00	44.63
5.00	80.00	13.00	86.76	160.00	0.00	49.63
5.00	85.00	12.50	91.63	160.00	0.00	54.63
5.00	90.00	12.00	96.44	160.00	0.00	59.63
5.00	95.00	11.50	101.20	160.00	0.00	64.63
5.00	100.00	11.00	105.90	160.00	0.00	69.63
5.00	105.00	10.50	110.54	160.00	0.00	74.63
5.00	110.00	10.00	115.14	160.00	0.00	79.63
5.00	115.00	9.50	119.67	160.00	0.00	84.63
5.00	120.00	9.00	124.16	160.00	0.00	89.63
5.00	125.00	8.50	128.60	160.00	0.00	94.63
5.00	130.00	8.00	132.98	160.00	0.00	99.63

```
$JOB            VARI,RUN=CHECK,TIME=30,PAGES=150
C
C       COMPUTER PROGRAM AND SAMPLE RESULTS FOR AN INVENTORY AND
C       ADVERTISING MODEL WITH TWO STATE VARIABLES BY STATE
C       INCREMENT DYNAMIC PROGRAMMING
C
        OLDS=-989898
        PI=50
        AN=150.2
        AB=70
        B=100
        C=2
        F=10
        CI=.15
        CA=1.5
        DX=5
        DQ=5
        DA=.2
        DT=.1
        NT=10
        NX=21
        NA=31
        NQ=30
        XMAX=100
        NQQ=NQ+1
        NXX=NX+1
        LINENO=0
        NBLOCK=0
        NS=NT
  100 FORMAT (1H1'STAGE NO   TIME      X(N-1)     Q(N-1)      X(N)
     1 Q(N)       P(N)       A(N)       PROFIT')
  101 FORMAT (1H ,5X,I2,8(4X,F7.2))
  102 FORMAT (1H '0BLOCK NO = ',I3)
        DIMENSION SB(51,51),SP(51,51)
        PRINT 100
        DO 71 K=1,51
        DO 71 L=1,51
        SB(K,L)=0
   71   SP(K,L)=0
        DO 10 KX=1,NX
        X=NX-KX
        X=X*DX
        DO 11 KQ=1,NQ
        Q=NQQ-KQ
        IQ=Q
        Q=Q*DQ
        T=1
        P=AB+B*(T-DT)
        NBLOCK=NBLOCK+1
        PRINT 102,NBLOCK
        DO 25 KA=1,NA
        A=KA-1
        A=A*DA
        XO=X+(P-Q)*DT
        QO=(Q*(1.+(C+A)*DT))/(1.+Q*(C+A)*DT/AN)
        IF (XO.LT..001) GO TO 25
        SNEW=(Q*F-(PI-X)**2*CI-CA*A*Q)*DT
        IF (SNEW.LT.OLDS) GO TO 25
        OLDS=SNEW
        AA=A
```

```
      XX=XO
      QQ=QO
   25 CONTINUE
      IF (LINENO.LT.45) GO TO 80
      LINENO=0
      PRINT 100
   80 PRINT 101,NS,T,X,Q,XX,QQ,P,AA,OLDS
      LINENO=LINENO+1
      SP(NS,IQ)=OLDS
   42 OLDS=-989898
      NS=NS-1
      T=NS*DT
      P=AB+B*(T-DT)
      DO 22 KA=1,NA
      A=KA-1
      A=A*DA
      U=P-Q
      IF (U.EQ..0) GO TO 90
      DELTX=ABS(DX/(P-Q))
      GO TO 91
   90 DELTX=999
   91 DELTQ=ABS(DQ/(Q*(C+A)*(1.-(Q+DQ)/AN)))
      IF (DELTX.GT.DELTQ) GO TO 51
      DELT=DELTX
      DELQ=DELT*Q*(C+A)*(1.-Q/AN)
      XO=X+(P-Q)*DT
      QO=(Q*(1.+(C+A)*DT))/(1.+Q*(C+A)*DT/AN)
      S1=(Q*F-(PI-X)**2*CI-CA*A*Q)*DELT
      Q1=SB(NS+1,IQ)+(SB(NS+1,IQ+1)-SB(NS+1,IQ))*DELQ/DQ
      Q2=SB(NS+2,IQ)+(SB(NS+2,IQ+1)-SB(NS+2,IQ))*DELQ/DQ
      S2=Q1-(Q1-Q2)*(DELT-DT)/DT
      SNEW=S1+S2
      GO TO 52
   51 DELT=DELTQ
      DELX=DELT*(P-Q)
      XO=X+(P-Q)*DT
      QO=(Q*(1.+(C+A)*DT))/(1.+Q*(C+A)*DT/AN)
      S1=(Q*F-(PI-X)**2*CI-CA*A*Q)*DELT
      X1=SP(NS+1,IQ+1)+(SB(NS+1,IQ+1)-SP(NS+1,IQ+1))*DELX/DX
      X2=SP(NS+2,IQ+1)+(SB(NS+2,IQ+1)-SP(NS+2,IQ+1))*DELX/DX
      S2=X1-(X1-X2)*(DELT-DT)/DT
      SNEW=S1+S2
   52 IF(SNEW.LT.OLDS) GO TO 22
      OLDS=SNEW
      AA=A
      XX=XO
      QQ=QO
   22 CONTINUE
      IF (LINENO.LT.45) GO TO 81
      LINENO=0
      PRINT 100
   81 PRINT 101,NS,T,X,Q,XX,QQ,P,AA,OLDS
      LINENO=LINENO+1
      SP(NS,IQ)=OLDS
      IF (NS.LE.1) GO TO 43
      GO TO 42
   43 T=1
      NS=NT
      OLDS=-989898
   11 CONTINUE
```

```
      DO 31 LNS=1,NT
      DO 31 LQ=1,NQ
   31 SB(LNS,LQ)=SP(LNS,LQ)
      T=1
      NS=NT
      OLDS=-989898
   10 CONTINUE
      STOP
      END
```

STAGE NO	TIME	X(N-1)	Q(N-1)	X(N)	Q(N)	P(N)	A(
BLOCK NO = 1							
10	1.00	100.00	150.00	101.00	150.03	160.00	
9	0.90	100.00	150.00	100.00	150.03	150.00	
8	0.80	100.00	150.00	99.00	150.03	140.00	
7	0.70	100.00	150.00	98.00	150.03	130.00	
6	0.60	100.00	150.00	97.00	150.03	120.00	
5	0.50	100.00	150.00	96.00	150.03	110.00	
4	0.40	100.00	150.00	95.00	150.03	100.00	
3	0.30	100.00	150.00	94.00	150.03	90.00	
2	0.20	100.00	150.00	93.00	150.03	80.00	
1	0.10	100.00	150.00	92.00	150.03	70.00	
BLOCK NO = 2							
10	1.00	100.00	145.00	101.50	145.84	160.00	
9	0.90	100.00	145.00	100.50	145.84	150.00	
8	0.80	100.00	145.00	99.50	145.84	140.00	
7	0.70	100.00	145.00	98.50	145.84	130.00	
6	0.60	100.00	145.00	97.50	145.84	120.00	
5	0.50	100.00	145.00	96.50	145.84	110.00	
4	0.40	100.00	145.00	95.50	145.84	100.00	
3	0.30	100.00	145.00	94.50	145.84	90.00	
2	0.20	100.00	145.00	93.50	145.84	80.00	
1	0.10	100.00	145.00	92.50	145.84	70.00	
BLOCK NO = 3							
10	1.00	100.00	140.00	102.00	141.60	160.00	
9	0.90	100.00	140.00	101.00	141.60	150.00	
8	0.80	100.00	140.00	100.00	144.36	140.00	
7	0.70	100.00	140.00	99.00	141.74	130.00	
6	0.60	100.00	140.00	98.00	142.87	120.00	
5	0.50	100.00	140.00	97.00	143.74	110.00	
4	0.40	100.00	140.00	96.00	141.60	100.00	
3	0.30	100.00	140.00	95.00	141.60	90.00	
2	0.20	100.00	140.00	94.00	141.60	80.00	
1	0.10	100.00	140.00	93.00	141.60	70.00	
BLOCK NO = 4							
10	1.00	100.00	135.00	102.50	137.32	160.00	
9	0.90	100.00	135.00	101.50	137.32	150.00	
8	0.80	100.00	135.00	100.50	141.36	140.00	
7	0.70	100.00	135.00	99.50	141.36	130.00	
6	0.60	100.00	135.00	98.50	141.36	120.00	
5	0.50	100.00	135.00	97.50	138.06	110.00	
4	0.40	100.00	135.00	96.50	139.02	100.00	
3	0.30	100.00	135.00	95.50	139.71	90.00	
2	0.20	100.00	135.00	94.50	140.33	80.00	
1	0.10	100.00	135.00	93.50	140.97	70.00	
BLOCK NO = 5							
10	1.00	100.00	130.00	103.00	132.98	160.00	
9	0.90	100.00	130.00	102.00	132.98	150.00	
8	0.80	100.00	130.00	101.00	138.26	140.00	
7	0.70	100.00	130.00	100.00	138.26	130.00	
6	0.60	100.00	130.00	99.00	137.62	120.00	

	Q(N)	P(N)	A(N)	PROFIT
.00	150.03	160.00	0.00	112.50
.00	150.03	150.00	0.00	586.73
.00	150.03	140.00	0.00	562.50
.00	150.03	130.00	0.00	281.25
.00	150.03	120.00	0.00	187.50
.00	150.03	110.00	0.00	140.63
.00	150.03	100.00	0.00	112.50
.00	150.03	90.00	0.00	93.75
.00	150.03	80.00	0.00	80.36
.00	150.03	70.00	0.00	70.31
.50	145.84	160.00	0.00	107.50
.50	145.84	150.00	0.00	1075.00
.50	145.84	140.00	0.00	1075.00
.50	145.84	130.00	0.00	358.33
.50	145.84	120.00	0.00	215.00
.50	145.84	110.00	0.00	153.57
.50	145.84	100.00	0.00	119.44
.50	145.84	90.00	0.00	97.73
.50	145.84	80.00	0.00	82.69
.50	145.84	70.00	0.00	71.67
.00	141.60	160.00	0.00	102.50
.00	141.60	150.00	0.00	512.50
.00	144.36	140.00	6.00	764.61
.00	141.74	130.00	0.20	2544.05
.00	142.87	120.00	2.20	2917.56
.00	143.74	110.00	4.20	643.58
.00	141.60	100.00	0.00	128.13
.00	141.60	90.00	0.00	102.50
.00	141.60	80.00	0.00	85.42
.00	141.60	70.00	0.00	73.21
.50	137.32	160.00	0.00	97.50
.50	137.32	150.00	0.00	252.32
.50	141.36	140.00	6.00	582.79
.50	141.36	130.00	6.00	886.08
.50	141.36	120.00	6.00	3730.15
.50	138.06	110.00	0.80	5218.54
.50	139.02	100.00	2.00	2950.93
.50	139.71	90.00	3.00	386.72
.50	140.33	80.00	4.00	215.32
.50	140.97	70.00	5.20	155.40
.00	132.98	160.00	0.00	92.50
.00	132.98	150.00	0.00	178.11
.00	138.26	140.00	6.00	290.26
.00	138.26	130.00	6.00	744.62
.00	137.62	120.00	5.00	1133.26

END

OF

OVERSIZED

DOCUMENTS

APPLICATION OF STATE INCREMENT DYNAMIC PROGRAMMING TO INDUSTRIAL MANAGEMENT SYSTEMS

by

RAMA NAND KAPUR

B.Tech.(Honors), Indian Institute of Technology
Kharagpur, India, 1957

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Industrial Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1969

One of the most important techniques developed in recent years for the solution of optimization problems is Bellman's dynamic programming. This technique solves at least in principle a large number of important optimization problems. However, because of the extremely large computational requirements, called by Bellman "the curse of dimensionality", only relatively simple problems can be solved on existing computers.

Various techniques have been suggested to overcome the dimensionality difficulty such as the polynomial approximation, lagrangian multipliers and ratio of state variables. However, none of these techniques are entirely satisfactory.

In 1965, a new technique called the state increment dynamic programming was developed by Larson. This technique requires considerably less fast-access memory than conventional dynamic programming but it still retains the general applicability and other desirable features of the conventional method. One important restriction is that this technique is applicable only to continuous processes.

The purpose of this report is to apply the new technique to industrial management systems and make a critical analysis of the results obtained and the computational procedure used.

First a brief review of conventional dynamic programming is made. The new technique is then described in some details. Finally, three models from the production planning area are considered. The first model involves a discrete case and serves mainly to illustrate a situation which cannot be treated by

the new technique. In the second model, production and inventory costs are minimized. The third model treats production and inventory control with advertising. From the solutions obtained by conventional dynamic programming and the new technique, the advantages as well as the disadvantages of state increment dynamic programming are highlighted. A critical comparison of the results is also made.