OPTIMIZED PRODUCTION PLANNING FOR ENERGY MANAGEMENT

by

STUART THOMAS CRAIG

B.Sc., Queen's University, Kingston, 1980

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

in
THE FACULTY OF GRADUATE STUDIES
Department of Electrical Engineering

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

June 1982

© Stuart Thomas Craig, 1982
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical Engineering

The University of British Columbia
1956 Main Mall
Vancouver, Canada
V6T 1Y3

Date June 8, 1982
ABSTRACT

A large proportion of the pulp and paper industry product cost is for energy. Increases in the cost of energy have led to energy conservation and energy management in mills. Energy costs can be reduced by scheduling production in such a way that demand charges for purchased electrical power are avoided, and by loading boilers in an efficient manner. A production planning method is presented that reduces energy costs by appropriately scheduling the operation of production units. The schedules are optimized by a multi-pass, successive approximations, variation of dynamic programming.

The optimization program is designed with pulp and paper mills as the target application, but it applies to other mills that can be modelled as a first order dynamic system of process units, interconnected by storage units.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>ii</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>viii</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. ALGORITHM SELECTION</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Possible Algorithms</td>
<td>10</td>
</tr>
<tr>
<td>2.1.1 Gradient Methods</td>
<td>10</td>
</tr>
<tr>
<td>2.1.2 Linear Programming</td>
<td>14</td>
</tr>
<tr>
<td>2.1.3 Dynamic Programming</td>
<td>17</td>
</tr>
<tr>
<td>2.2 Choosing an Algorithm</td>
<td>26</td>
</tr>
<tr>
<td>2.3 Problem Formulation for Dynamic Programming</td>
<td>28</td>
</tr>
<tr>
<td>2.4 Unanticipated Problems</td>
<td>35</td>
</tr>
<tr>
<td>2.5 Conclusions</td>
<td>36</td>
</tr>
<tr>
<td>3. THE COST FUNCTION AND PROGRAM PARAMETERS</td>
<td>38</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>38</td>
</tr>
<tr>
<td>3.2 Implementation of the Desired Cost Factors</td>
<td>39</td>
</tr>
<tr>
<td>3.2.1 Factors Handled by Constraints</td>
<td>39</td>
</tr>
<tr>
<td>3.2.2 Material Balances and Process Dynamics</td>
<td>41</td>
</tr>
<tr>
<td>3.2.3 Bottleneck Departments</td>
<td>41</td>
</tr>
<tr>
<td>3.2.4 Planned Shutoffs</td>
<td>42</td>
</tr>
<tr>
<td>3.2.5 Process Rate Changes</td>
<td>43</td>
</tr>
<tr>
<td>3.3 Energy System Simplifications</td>
<td>44</td>
</tr>
<tr>
<td>3.3.1 Boiler Load Allocation</td>
<td>44</td>
</tr>
<tr>
<td>3.3.2 Mill Steam Systems</td>
<td>46</td>
</tr>
<tr>
<td>3.3.3 Turbogenerators</td>
<td>47</td>
</tr>
<tr>
<td>3.3.4 Grouping Power Sources</td>
<td>48</td>
</tr>
<tr>
<td>3.3.5 The Simplified Energy System</td>
<td>49</td>
</tr>
<tr>
<td>3.4 Parameter Selection</td>
<td>50</td>
</tr>
<tr>
<td>3.4.1 Number of Grid Points</td>
<td>52</td>
</tr>
<tr>
<td>3.4.2 Number of Iterations</td>
<td>53</td>
</tr>
<tr>
<td>3.4.3 Iteration Order</td>
<td>55</td>
</tr>
<tr>
<td>3.4.4 The Parameters Used</td>
<td>55</td>
</tr>
<tr>
<td>4. RESULTS</td>
<td>58</td>
</tr>
<tr>
<td>4.1 Process Description</td>
<td>58</td>
</tr>
<tr>
<td>4.2 Program Verification</td>
<td>63</td>
</tr>
<tr>
<td>4.3 Savings Obtained</td>
<td>73</td>
</tr>
<tr>
<td>4.4 Impossible Schedules</td>
<td>79</td>
</tr>
<tr>
<td>4.5 Conclusions</td>
<td>80</td>
</tr>
</tbody>
</table>
LIST OF TABLES

2.1 Optimal Routes for the Example of Figure 2.1 ....... 19

2.2 Comparison of Memory Requirements for Direct Grid
Point Storage and Polynomial Approximation
Coefficient Storage ...................................... 22

4.1 Energy Generation and Usage in a Typical Pulp and
Paper Mill ..................................................... 64
LIST OF FIGURES

1.1 A Typical Pulp and Paper Mill Steam and Electrical System Configuration ........................................ 2
1.2 A Small, Five Process/Buffer Kraft Mill Model .......... 4
2.1 Optimal Routes for the Example of Figure 2.1 .......... 19
2.2 Cost Function Value Versus Iteration Number for the "One Buffer at a Time Method" .............................. 32
3.1 Steam Production, Turbogenerator Rate and Purchased Power Assignment Logic ................................. 51
4.1 A Multiple Loop, 8 Process Mill Model .................. 59
4.2 A Twelve Process Mill Model .............................. 60
4.3 Erratic Output Production ................................ 66
4.4 Planned Shutoff of P2 During Times 6, 7 ............... 69
4.5 Breakdown of P1 During Times 1, 2 ..................... 71
4.6 Breakdown of P9 During Times 1, 2, and A Planned Shutoff of P3 During Times 8, 9 ....................... 74
4.7 Purchased Power Billing Rate Increase at Time 7 .. 77
5.1 Photograph of Graphics Terminal Display During the Input of Buffer Level Constraints ........................ 84
5.2 Colour Display of Optimal Trajectories Within a Small Mill Schematic ........................................... 84
A1 Flowchart Symbols ........................................... 96
A2 Successive Approximations, Multi-pass Dynamic Programming, Main Program ............................... 97
A3 Dynamic Programming Subroutine (DP) ................... 98
B1 Computational Work Index versus Initial Grid Spacing and Number of Grid Points, and, Constraint Closure Rate versus Initial Grid Spacing and Number of Grid Points ........................... 111
C1 Graphical Input Program (Main Program) ............... 113
C2 Graphical Input Program (Subroutine GIP) ............. 115
C3 Graphical Input Program (Subroutine CURIN) ....... 115
<table>
<thead>
<tr>
<th>D1</th>
<th>Graphical Output Program</th>
<th>124</th>
</tr>
</thead>
</table>

vii
ACKNOWLEDGEMENTS

The helpful suggestions and assistance throughout the thesis, of Dr. M. S. Davies are gratefully acknowledged. I also thank the MacMillan Bloedel Limited for their cooperation, and the Science Council of British Columbia and the Natural Sciences and Engineering Research Council of Canada for their support.
1. INTRODUCTION

The pulp and paper industry is energy dependent. In 1978, an average of 15% of the industry's product cost was attributed to energy. The significance of this energy demand is emphasized by the pulp and paper industry's 18% share of the total Canadian industrial energy consumption [DIC81]. Rising fuel costs have provided incentives for attempts to reduce the industry's energy bill. This bill can be reduced by converting existing equipment or by adding new equipment and so enabling cheaper fuel to be used. Conservation measures can be used to reduce the amount of energy needed. Improving the efficiency of mill equipment, in order to reduce the energy demand, and managing the mill in such a way as to avoid power demand charges, time-of-day charges and inefficient steam plant operation also reduces the energy costs. Rate charges can account for up to 50% of the purchased energy bill; therefore, avoiding these charges represents a significant savings [HAN]. This thesis develops a procedure for reducing energy costs by careful management.

Typically, a large mill has a number of power boilers, recovery boilers and turbo-generators as well as a connection into an electrical utility's grid (see Figure 1.1). Power boilers burn: coal, oil, gas and/or hog fuel (bark and wood
Figure 1.1: A Typical Pulp and Paper Mill Steam and Electrical System Configuration
wastes). Recovery boilers are part of a material feedback loop in the kraft (chemical) pulping process (see Figure 1.2). The "cooking" chemicals (weak black liquor), used in breaking down wood into pulp, are concentrated in evaporators by water removal. The resulting strong black liquor contains organic material from the non-fibrous parts of wood (lignin), removed from the wood during pulping. The inorganic portions of the liquor are the cooking chemicals to be recovered. Heat from burning the organic portion of the liquor in a recovery boiler is used to make steam. Smelt, the inorganic remains of black liquor combustion, undergoes further chemical modification in the causticizing plant, to produce fresh cooking chemicals (white liquor) [GRA81]. Turbo-generators are used for the production of electricity and for reducing steam pressure to that required by the mill. Process steam not expanded by the turbo-generator is supplied by pressure reducing valves.

The energy needs of the different production units vary widely. Groundwood pulping involves grinding blocks of wood between stone wheels. If these wheels are driven by electrical motors, the groundwood plant would have large electrical and small steam demands. The kraft pulping section, on the other hand, would have a large steam demand for 'cooking' purposes. Paper machines employ steam heat to dry paper and are usually driven by electrical motors, but older paper machines may be driven by steam turbines. Each process thus has its own steam demand, with this demand to be met at high, intermediate, or low pressure levels. These diverse energy requirements must be
Figure 1.2: A Small, Five Process/Buffer Kraft Mill Model

Key:

- P1 - digester
- P2 - evaporator
- P3 - recovery boiler
- P4 - causticising plant
- P5 - bleach plant and paper machines
- T1 - unbleached kraft
- T2 - weak black liquor
- T3 - strong black liquor
- T4 - green liquor
- T5 - white liquor
satisfied from the energy sources available and at the same time, the desired production schedule must be met and the cost of providing this energy minimized.

Work has been done on the optimal allocation of boiler loads and turbo-generator settings so that the cost of supplying the energy demands of the mill is minimized. Various computational approaches have been employed, such as: linear programming [HUN80], the iterative assignment of small steam load increments to the currently cheapest boiler [HAN, LEF78], and optimum search algorithms [CHO80]. Although these methods decrease energy costs by meeting the demand as inexpensively as possible, nothing is done to optimize this demand so that, over time, the cost is minimized. This dynamic aspect of the problem has been treated and a number of optimization techniques have been applied to the problem, including linear programming, decomposed in time [PET69, TIN74]. Decomposition is required to reduce the problem to a practical size. The method is successful at reducing the process rate changes and scheduling a planned shutoff. A disadvantage resulting from the time decomposition is that the type of cost function to be minimized is restricted. Attempts to reduce energy costs concentrated on smoothing the purchased power demand over time, as this is possible with the cost function restrictions. Smoothing energy demands does not take advantage of time-of-day charges. Certain parts of the plant are ignored, as material feedback loops are uncontrollable. This can lead to empty or overflowing storage tanks if the calculated "optimum"
schedule is followed. This thesis presents a specific method of solving the dynamic scheduling problem that does not have the above mentioned shortcomings.

With the rapid advances in computer technology, the question arises: to what extent should control be relinquished to computers? Ergonomic studies on this question have led to computer based supervisory control systems that have information display facilities with monitoring, planning and optimization aids for operators [JAN81]. The concept of using the computer as a control "aid" is taken in this thesis. Monitoring and planning aids have an estimated cost of $300,000 to $1.5 million (1980 dollars) with a predicted return from energy cost reductions and production increases of $300,000 to $2 million per year [AAR80].

For the purposes of supervisory planning, the mill is simplified and modelled as production units interconnected by storage units. This type of model has been used in previous studies on optimal scheduling [PET69, TIN74, URO80]. The accuracy of the model was found to be sufficient; however, constant updating and correcting of the model parameters was advised. A production unit controls the flow of materials between storage units, and may consume/produce energy. A 'process' may be as small as a valve or as large as a number of paper machines. Storage units allow flexibility in production rates between the different sections of the mill. Peaks in energy demand can be reduced by making use of this
flexibility. Using the simplified mill model, a computer program is used to optimize the production rates, given: planned process shutdowns, desired final product production rates, storage capacity constraints and process rate limits. Although the program is intended to apply to pulp and paper mills, the type of model used applies to many "inventory" type of problems.

A variation of dynamic programming was chosen as the optimization method [BEL57, BEL62, LAR67]. This algorithm is used because of the relative ease with which constraints are handled as compared to other methods [SAG68]. A mill scheduling problem is subject to storage buffer capacity constraints, production rate constraints, and desired production schedules. Inclusion of many constraints is necessary in order to obtain a useful solution. A serious draw-back of dynamic programming is the "curse of dimensionality". The problem size is exponentially related to the number of states (storage units). Even a small, 10 unit model would exceed the capacity of today's computers. The modifications to dynamic programming used are the successive approximations and multi-pass variations [LAR70, BEC77]. These techniques are used because the drastic reduction obtained in the dimensionality handicap of dynamic programming allows more realistically dimensioned problems to be solved.

The thesis will deal with the choice of optimization algorithm, refinements to the algorithm, cost function
considerations, test results and a description of graphical input/output routines used to facilitate the operation of the main program.
2. ALGORITHM SELECTION

Given the degree of detail required from the mill model, and the desire to minimize energy costs and production disruptions, the next step in solving the problem is to find an algorithm that is suitable and efficient. An important aspect of the production scheduling problem is the large number of constraints. Buffers must neither go dry nor overflow, and processes are limited to operate within certain ranges. Planned process shutoffs and the desired final product production rates must be scheduled, adding further constraints to the problem.

Product production schedules are supplied to the optimization program from information on customer orders and shipping schedules. This production plan attempts to reduce product grade changes while meeting customer demands [URO80]. Since this stage of planning is a separate problem, dependent on separate data (such as market conditions), the planning aid presented here does not attempt to optimize the final product production. It is assumed that mill management has determined how and when they are going to fill customer orders. This information is then supplied to the energy management program in the form of additional constraints.
2.1 Possible Algorithms

Three optimization methods were considered: the steepest descent (gradient) method, linear programming and its variations, and dynamic programming. Leiviska and Uronen have used a variation of the gradient method, Tamura's algorithm, to solve the pulp and paper mill scheduling problem [URO80]. Pettersson [PET69] and Tinnis [TIN74] used a linearized mill performance index to produce production schedules with linear programming. Dynamic programming has been used to solve scheduling problems for power generation [ROS80, BEC77], water reservoir usage, and airline flight scheduling [LAR70, LAR67].

2.1.1 Gradient Methods

The steepest descent method is based on making small adjustments to the problem variables in the direction which provides the maximum reduction in the cost function. This direction is found by determining the gradient of the cost function with respect to the problem variables. Problem constraints are included with the aid of Lagrange multipliers. The dynamic aspects of a system are handled by using a Hamiltonian function [SAG68].

Consider the simple example:
$$\phi = x_1^2 + 2x_2^2, \quad g(x)=5-x_1-x_2 = 0,$$
where $\phi$ is the cost function to be minimized and $g(x)$ is the constraint. We use:
\[ \dot{\phi}_x + \lambda g_x = 0, \]
where \( \phi_x \) is the gradient of \( \phi \) with respect to \( x \) and \( g_x \) is the gradient of \( g \) with respect to \( x \). \( \lambda \) is a scalar Lagrange multiplier. This yields:

\[
\begin{bmatrix}
2x_1 \\
4x_2
\end{bmatrix} + \lambda
\begin{bmatrix}
-1 \\
-1
\end{bmatrix} = 0; \\
therefore, \quad \lambda = \frac{20}{3}
\]

\[ \begin{bmatrix}
20/3 \\
20/3
\end{bmatrix} \]

\[ x_1 = \frac{10}{3}, \quad x_2 = \frac{5}{3}. \]

Now consider a very simple dynamic system:

\[ \dot{x} = f(x, u) \]
\[ \dot{x}_1(t) = u(t) \]
\[ \dot{x}_2(t) = (1-x_1(t))^2 \]
\[ g(x) = x_1(1) - 1 = 0 \]

\[ \phi = x_2(1) \]

\[ g_x = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \phi_x = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \]

Define the Hamiltonian, \( H \), as:

\[ H = pf \]

where \( p \) is the costate vector (Lagrange multipliers) and,

\[ \dot{p} = -H_x \]

If we try \( u(t)=1, \quad 0 \leq t \leq 1 \) as an initial guess, then:

\[ x_1(t)=t, \quad x_1(1)=1, \quad g=0, \]
\[ x_2(t)=(1-t)^2, \quad x_2(t)=(t^3/3)-t^2+t, \quad x_2(1)=1/3=\phi. \]

Improvements in the solution are accomplished by making moves, \( \delta u_2 \), into the constraint region \( (g(x)=0) \), then making moves to reduce \( \phi, \delta u_1 \). These moves are functions of \( H_u, \phi_u, g_u, \) and \( \bar{g} \).
where the subscript \( u \) refers to the differential with respect to the control \( u \) and \( \bar{g} \) is the value of \( g(x) \) for the current solution values of \( x \). Since to start, \( \bar{g} = 0 \), the constraint is met and a \( \delta u \), move may be made. Arbitrarily choosing a constraint step size constant equal to 1 gives:

\[
\begin{align*}
  u(t) &= t^2 - 2t + 7/6 \\
  x_1(t) &= (t^3/3) - t^2 + 7t/6 \\
  \bar{g} &= -0.5
\end{align*}
\]

Now, \( \bar{g} \neq 0 \), so a \( \delta u \) move is made (with the correction step size constant equal to 1) giving:

\[
\begin{align*}
  u(t) &= t^2 - 2t + 5/3 \\
  x_1(t) &= (t^3/3) - t^2 + 5t/3, \quad \bar{g} = 0 \\
  x_2(1) &= .2529, \text{ about } 24\% \text{ less than the initial cost.}
\end{align*}
\]

Another iteration gives \( \bar{g} = 0 \) and \( \phi = .200 \), which is 40% less than the initial cost. The iterations are continued until improvements in \( \phi \) are insignificant.

The major disadvantages of the steepest descent method are the added complications resulting from constraints, and the amount of numerical integration required. The constraints of the scheduling problem are functions of time, not just end point constraints as used in the previous example. These constraints can be included by adding another state, \( x_{n+1} \), to the problem:
$x_{n+1}(t) = Hx_1 + Hx_2 + \ldots + Hx_n + Hx_m + Hu_1 + Hu_2 + \ldots + Hu_m$, where:

- $n$ = the number of states in the original problem,
- $m$ = the number of controls in the original problem,
- $Hx_i(t) = 1$ if buffer $i$ is dry at time $t$
  0 otherwise
- $Hx_{i+1}(t) = 1$ if buffer $i$ is overflowing at time $t$
  0 otherwise

$Hu_i$ is as $Hx_i$, but for process rates.

This new state is then added to the cost function $\psi$, or used with an endpoint constraint:

$$g(x) = x_{n+1}(\text{end time}) = 0$$

The problem is significantly complicated by this additional state and may become ill-conditioned as a result of its addition [TAM75].

Tamura developed an algorithm, based on the gradient method, that handles time dependent constraints [TAM75]. This algorithm breaks the problem into a series of smaller, one time stage problems. Since the problems have only one time stage, constraints are made up of initial conditions and endpoint constraints. Decomposition in time is possible because the optimization is done with a fixed $p$ trajectory; then, after an iteration of optimizing with respect to $x$ and $u$, $p$ is updated using the appropriate gradient from a dual formulation of the problem. Leiviska found Tamura's algorithm to have "short" execution times and "moderate" memory requirements. Although the algorithm is particularly suited to
higher order (distributed-lag) systems, it has been used by Leiviska and Uronen with the first order mill model and a quadratic-linear cost function [URO80]. For the Tamura algorithm, optimizing over each time step constitutes the majority of the computations. In order to be efficient, these optimizations are carried out by a guided search method such as the gradient method.

2.1.2 Linear Programming

Linear programming and its variations have been used to solve the pulp and paper mill production scheduling problem [PET69, TIN74, URO80]. Linear programming solves problems of the form:

$$\begin{align*}
\text{minimize } & \quad z, \quad z = cx, \\
\text{subject to: } & \quad Ax = b, \\
& \quad x > 0.
\end{align*}$$

For example:

$$\begin{align*}
\text{minimize: } & \quad z = x_1 + 2x_2 + 3x_3, \\
\text{subject to: } & \quad x_1 \leq 1, \\
& \quad x_2 \leq 3, \\
& \quad x_3 \leq 2, \\
& \quad x_1 + x_2 + x_3 = 5, \\
& \quad x > 0.
\end{align*}$$

Inequality constraints are converted to equality constraints by the addition of "slack" variables. In this case, $x_4$, $x_5$, and $x_6$ are the slacks:
x_1 + x_6 = 1
x_2 + x_6 = 3
x_3 + x_6 = 2
x_1 + x_2 + x_3 = 5
x \geq 0.

Linear programming is based on the fact that the minimum value of z occurs at an intersection of constraints. In some cases an equally small value of z can exist at a number of constraint intersections and/or within the constrained region, but the first intersection found that produces the minimum z, is the only result produced. The basic operation of the linear programming algorithm is to move the solution point along the constraint boundary in the direction of maximum reduction in z. When no improvement in z is possible, the minimum has been found.

Time dependence is managed by having a set of variables for each discrete time step. A small example of a portion of the A matrix for a dynamic system is given. Only one of the variables is shown.

For: \[ x_3(k+1) = x_3(k) + u_5(k) - u_6(k), \quad k = 0, 1, 2, 3 \]

\[ 0 \leq x_3 \leq 1, \text{ all } k, \]
\[ 0 \leq u_5 \leq 1, \text{ all } k, \]
\[ 0 \leq u_6 \leq 1, \text{ all } k; \]

the linear programming constraints would be:

\[ x_3(1) + \text{slack1} = 1 \]
\[ x_3(2) + \text{slack2} = 1 \quad 0 \leq x_3 \leq 1 \]
\[ x_3(3) + \text{slack3} = 1 \]
If this approach were extended to a model having 10 buffers, 11 processes (with both upper and lower limits) which is to be optimized over 20 time periods, the A matrix becomes 1040 x 1260 in size. 840 (67%) of the columns are for slack variables. Such a large problem is impractical as a method of finding the optimal schedule. Pettersson [PET69] decomposed the problem in time so that a time period would consist of a linear programming problem of 50 x 40 columns, still a medium sized problem. A "performance" index is passed from one time stage to the next, to link the stages into one problem. This approach was used, successfully, to reduce process fluctuations; hence, the performance index used was a measure of deviation from a predetermined average process rate. Tinnis attempted to reduce energy costs by including a performance index that measured the deviation from a predetermined average energy consumption rate [TIN74]. Billing rate changes would complicate the determination of these desired averages. With this method, it is not possible to specify all buffer levels for the end of the planning period. A storage tank in a feedback loop was not scheduled. It was assumed that the tank would not overflow, and that if extra material was needed it could be supplied from a source that is external to the model.
This problem of not being able to specify all buffer levels is explained further in section 2.3.

Recently, the "out-of-kilter" algorithm has been used for the scheduling problem by Leiviska and Uronen [URO080]. This algorithm is a modification of linear programming that formulates the problem as a network flow problem [BAZ77]. The "out-of-kilter" algorithm, as is the case with linear programming, has dimensionality problems and is restricted to linear cost functions. The "out-of-kilter" algorithm was found to have longer execution times than, and similar memory requirements to, Tamura's algorithm.

2.1.3 Dynamic Programming

Dynamic programming is a discrete optimization method that handles constraints easily and can be used with a wide variety of cost functions. Dynamic programming is an exhaustive search of all allowable solutions. It is more efficient than straight enumeration because no portion of a solution is considered more than once. This is accomplished by finding the solution backwards in time. Dynamic programming has been applied to: minimizing fuel usage by aircraft route planning, airline flight scheduling, water reservoir - power dam planning, network flow problems, missile interception, and optimizing electrical generation on a large power grid [ROS80, LAR67, BEC77].

As an example, consider the one-way traffic problem
shown in Figure 2.1 [KIR70]. A delivery boy must get to 'h' from any of 'a' through 'g', in the least amount of time. The time for each transition between nodes is shown beside the path between nodes. The dynamic programming approach to solving this problem is to start at 'h' and work backwards to the furthest point from 'h', 'a'. At each node, all possible routes are tried. A trial only involves adding the cost of one transition to the cumulative cost at the next node. This cumulative cost was calculated previously in the same way the current node's cumulative cost is calculated. The cheapest path is found from all the possibilities available at the current node. The cost of this path is made the cumulative cost for the node being considered. The route to the next node which provides this minimum cost is stored so that the best direction of travel will be known in the event that the delivery boy finds himself at the current node. The results of this dynamic programming procedure are shown in Table 2.1. From Table 2.1, the fastest route from 'a' to 'h' is: 'a', 'd', 'e', 'f', 'g', 'h', with a "cost" of 18 minutes.

The basic relationship of dynamic programming can be expressed as:

$$J(k,x(k)) = \min \{ $(u(k),x(k)) + J(k+1,f(u(k),x(k)) \},$$

over all $u(k)$

where the system is described by:

$$x(k+1) = f(x(k),u(k)),$$

the cost of control $u(k)$ from state $x(k)$ is: $(u(k),x(k))$ and $J(k,x(k))$ is the minimum cost possible at time $k$ and state
Figure 2.1: Street Map for the Delivery Boy Example

Table 2.1: Optimal Routes for the Example of Figure 2.1.

<table>
<thead>
<tr>
<th>Node</th>
<th>Next Node</th>
<th>Cost</th>
<th>Best Next Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>h</td>
<td>0+2=2</td>
<td>h</td>
</tr>
<tr>
<td>f</td>
<td>g</td>
<td>3+2=5</td>
<td>g</td>
</tr>
<tr>
<td>e</td>
<td>h</td>
<td>0+8=8</td>
<td>f</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td>3+7=10</td>
<td>e</td>
</tr>
<tr>
<td>c</td>
<td>f</td>
<td>3+5=8</td>
<td>f</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>9+8=17</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>5+17=22</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>8+10=18</td>
<td></td>
</tr>
</tbody>
</table>
\( x(k) \) [KIR70]. Bearing in mind this basic dynamic programming relationship, dynamic programming can be considered to be a series of single time step problems. At a time stage, the best control to apply at this time is found for every discrete state-value. Time intervals that follow are accounted for by the \( J(k+1, x(k)) \) term of the recursive, dynamic programming relationship. The cumulative cost for \( x(k) \), \( (J(k, x(k)) \), is found by adding the cost of applying the optimal control, and the cumulative cost associated with the resulting state, \( x(k+1) \). The problem over the preceding time interval is then solved by using these cumulative costs.

The delivery boy example is particularly simple because it does not demonstrate the problems of cost interpolation and dimensionality. Dynamic programming is a discrete algorithm, requiring the problem variables to be quantized to certain levels. For a quantized \( u(k) \) and \( x(k) \), the resulting \( x(k+1) \) may not be one of the quantized \( x(k+1) \) and hence, there would be no stored value for \( J(k+1, x(k+1)) \). This value would have to be interpolated from surrounding values. As the dimension of \( x \) becomes greater, this interpolation becomes significant. Quantization is also the source of what Bellman calls the "curse of dimensionality" [BEL57]. The amount of storage and computation required grows exponentially with the problem size. A model consisting of 10 buffers, 11 processes and 20 time stages which has buffer levels and process rates represented as 25 quantized levels (grid points), would require at least: \( 25^{10} = 9.5 \times 10^{13} \) high speed memory locations,
20\times 11 \times 25^{10} = 2.1 \times 10^{16} \text{ low speed memory locations and}\n20 \times 25^{10} \times 25^{11} \times T_c = 4.5 \times 10^{30} \times T_c \text{ seconds of CPU time. } T_c \text{ is the number of seconds required to calculate the cost of one transition from a specific } x(k) \text{ to one of the } x(k+1). \text{ If one were optimistic and chose } T_c = 10^{-6} \text{ seconds, the CPU time required is } 1.4 \times 10^{17} \text{ years. Because of the extreme memory and CPU time requirements of 'direct' dynamic programming, various modifications to the algorithm have been developed.}

Interpolation and dimensionality problems can be reduced by using polynomial interpolation of the cost function at each time stage. After finding the minimal costs at each grid point for a time stage, the coefficients for a polynomial approximation to these costs are found and stored. If the number of grid points and states is large, the time spent finding these coefficients should be more than offset by memory and interpolation time savings. For } n \text{ states, an approximation of degree } d \text{ would produce } (d+1)^n \text{ coefficients per time stage } [BOU71]. \text{ Table 2.2 shows the savings possible with this method for various } n, d \text{ and } l \text{ (the number of quantization levels). Despite these savings, significant memory would still be required for reasonably sized problems of 10 to 20 process units and a polynomial of degree 4 or greater.}

A technique called "state increment dynamic programming" reduces the computation and memory requirements of dynamic programming by using a variable time interval. It
Table 2.2: Comparison of Memory Requirements for Direct Grid Point Storage and Polynomial Approximation Coefficient Storage.

<table>
<thead>
<tr>
<th>n</th>
<th>l</th>
<th>R.M.R.</th>
<th>d</th>
<th>(1+d)**n</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10^1</td>
<td>10^5</td>
<td>2</td>
<td>243</td>
</tr>
<tr>
<td></td>
<td>10^2</td>
<td>10^10</td>
<td>4</td>
<td>3125</td>
</tr>
<tr>
<td></td>
<td>10^3</td>
<td>10^15</td>
<td>8</td>
<td>59049</td>
</tr>
<tr>
<td></td>
<td>10^4</td>
<td>10^20</td>
<td>9</td>
<td>10^5</td>
</tr>
<tr>
<td>10</td>
<td>10^1</td>
<td>10^10</td>
<td>2</td>
<td>59049</td>
</tr>
<tr>
<td></td>
<td>10^2</td>
<td>10^20</td>
<td>4</td>
<td>9.77x10^6</td>
</tr>
<tr>
<td></td>
<td>10^3</td>
<td>10^30</td>
<td>8</td>
<td>3.49x10^9</td>
</tr>
<tr>
<td></td>
<td>10^4</td>
<td>10^40</td>
<td>9</td>
<td>10^10</td>
</tr>
<tr>
<td>20</td>
<td>10^1</td>
<td>10^20</td>
<td>2</td>
<td>3.49x10^9</td>
</tr>
<tr>
<td></td>
<td>10^2</td>
<td>10^40</td>
<td>4</td>
<td>9.54x10^13</td>
</tr>
<tr>
<td></td>
<td>10^3</td>
<td>10^60</td>
<td>8</td>
<td>1.22x10^19</td>
</tr>
<tr>
<td></td>
<td>10^4</td>
<td>10^80</td>
<td>9</td>
<td>10^20</td>
</tr>
</tbody>
</table>

where:
- n = number of states (buffers),
- l = number of quantization levels,
- R.M.R = raw memory requirement (the memory required to store cost information without polynomial approximation),
- d = degree of the polynomial to be used for approximating cost information,
- (1+d)**n = memory requirements for cost information when polynomial approximation is used.
has been used by Larson to optimize SST flight paths in order to minimize fuel costs [LAR67]. The time interval used is the minimum time required to produce a specified incremental change in a state. This is a method of incorporating sampling interval sensitivity with discrete dynamic programming [SAG68]. The problem is then split into blocks of $w_i$ state increments by $\Delta T$ time units. The subscript 'i' refers to state number 'i'. A block can have a different number of increments for each state variable. These problem blocks are then solved one at a time, with special consideration being given to the block boundaries so that the trajectories can pass from one block to the next. High speed storage requirements are reduced because of the small dimensionality of the problem blocks. The number of state increments per block is typically 4 to 6 and the time length of a block, $S$, is 5 to 10 time increments.

The high speed memory requirement (HSMR) of state increment dynamic programming is given by:

$$
HSMR = 2\sum_{i=1}^{n} w_i + S\left[\sum_{i=1}^{n} (w_i + 1) - \sum_{i=1}^{n} w_i\right],
$$

where $w_i$ and $S$ are as described above, and $n$ is the number of states in the model.

As an example of the memory savings possible, consider a 15 state model with 40 quantization levels per time stage. If $w_i$ is 4 for all $i$ and $S$ equals 7, then $HSMR = 2.1 \times 10^{11}$, which is still a large amount of memory. 'Direct' dynamic programming would need about $40^{15} = 1.1 \times 10^{24}$ locations for this example.
The search time required to find the best cost and best control at each state grid point can be reduced in some cases by using an aided search [LAR67]. Instead of exhaustively comparing the costs of all possible controls, a method such as linear programming, for instance, could be used to find the optimal trajectories from one time stage to the next. This approach is useful only when the number of states and quantization levels is large compared to the added computational complexity of using a method more sophisticated than a direct search. Directed search methods restrict the cost function to a form that meets the requirements of the method. Linear programming would require a linearized cost. The steepest descent method would need the gradient of the cost function.

Multi-pass dynamic programming reduces the problem size by using a small number of grid points. After solving the optimization problem using these few, coarse, grid points, the increments between grid points are reduced and the procedure repeated until the grid points are sufficiently close together [BEC77]. Not only are memory requirements reduced due to the fewer grid points, but computation time can be reduced because of the fewer cost calculations and comparisons required. The range of values under consideration do not cover the complete allowable range for the states as there are too few grid points. As a result of this, in some poorly 'behaved' problems, convergence may not be to the true optimum. For instance, convergence may be to a local minimum for cases
where the cost function is very sensitive to small changes in a variable which produces a deep, narrow 'trough' in the cost. The initial coarse grid may not include the 'trough' and so the solution is moved away from the global optimum. When the grid spacing is small enough to resolve the sharp dip in the cost function, the solution trajectory may be too far from the true minimum to include it as a possible trajectory. Another example of the multi-pass method failing is when constraints are closed in around the solution trajectory too quickly, restricting the final solution to be close to the initial, rough, starting trajectory.

Successive approximations achieves substantial reductions in both computation time and memory requirements by optimizing about one variable at a time, with the other independent variables being held at their 'old' values. A number of iterations must be made to obtain convergence because the solution with respect to variable x may change after the solution with respect to variable y is found. Convergence to the optimum with this algorithm has been proven for only a few types of problems [LAR70]. There are many cases where this algorithm cannot converge or may converge to a local minimum (e.g., see the 'one-buffer-at-a-time' method of section 2.3). Problems that have been successfully solved with this method include unconstrained problems with quadratic cost functions, optimal planning of system additions (e.g., power generator addition to a power grid), optimizing multi-purpose, multi-reservoir water systems, and scheduling a fleet of
aircraft over a fixed set of routes. Successive approximations has been used, for example, to solve the aircraft scheduling problem where 100 planes were involved. The water system problem resembles the mill production scheduling problem, since the process consists of dams which are interconnected by reservoirs (buffers). The flow rates through the dams cannot exceed a maximum or be negative; reservoir levels cannot overflow or go dry, just as process rates and buffer levels are constrained.

2.2 Choosing an Algorithm

In review, the methods considered were the gradient method, linear programming, and dynamic programming. Single time stage problems were solved using linear and quadratic programming packages available from the UBC Computing Centre. These trials revealed that the methods would require modest CPU time but substantial memory. A rough estimate for a 10 buffer, 10 process, 20 time stage model, shows that 10 seconds of CPU time and 3.2M bytes of memory would be required [PAT80]. Quadratic programming is similar to linear programming except that the cost function may be a quadratic.

Problems associated with the large number of double sided, time dependent, inequality constraints discouraged the pursuit of the steepest descent method. Dynamic programming is attractive because of the ease with which constraints and cost functions can be handled. Constraints are accounted for by
simply not considering grid points that fall outside the allowable region. In fact, the more tightly constrained the problem, the less computation required since fewer grid points need be considered. Practically any cost function may be used since dynamic programming deals with the value of the cost function at a state and time; the gradient of the cost function is not needed.

Dynamic programming is also attractive from a programming point of view. The concept of the algorithm is simple and does not require information on gradients of cost functions, and penalty functions as does the Tamura algorithm.

If dynamic programming is to be used, the "curse of dimensionality" must be overcome. Of the methods described above, cost interpolation, state increment dynamic programming and the use of search procedures, would be of limited value since the accuracy of the model, state measurements and mill controls does not warrant the use of a fine grid (large number of quantization levels). The variable time step aspect of the state increment method would be useless as the simplicity of the model would cause all the time steps to be the same. This is because an incremental change in a state variable is brought about by the same process rate - time interval combination at all times and states of the problem. Successive approximations provides savings in memory and CPU time needs, independent of the number of quantization levels used. As mentioned, successive approximations has been used
successfully with a problem similar to the mill scheduling problem. For these reasons the successive approximations method was selected as a good way to make dynamic programming feasible.

The mill model is a first order dynamic linear system; hence, one would expect the multi-pass approach would not hinder convergence. A combination of successive approximations and the multi-pass method would produce an algorithm having modest memory and CPU time requirements. This combination is the solution method used in this thesis.

2.3 Problem Formulation for Dynamic Programming

The details of the successive approximations, multi-pass dynamic programming algorithm are influenced by the choice of independent variables, and by the way these independent variables are related to the remaining dependent variables. Optimization may be carried out either with respect to buffers levels or process rates. If buffer levels are used as the independent variables, a change in buffer levels would imply a specific set of process rates. By considering transitions from one quantized buffer level to another quantized buffer level at the next time stage, no cost interpolation would be required. All transitions would end at a grid point, the cost of these transitions being a function of the change in buffer levels. Buffer level constraints are handled easily because transitions to disallowed levels would
never be considered. Process rate constraints could be accounted for by penalizing any transition requiring an out of bounds process rate. A problem with this approach is that arbitrary transitions in buffer levels cannot be specified for models containing feedback/feedforward loops. Buffer levels cannot be specified in loops because these process loops are uncontrollable. From a practical point of view, this is because both the loop input and loop output conditions must be satisfied. The change in the total amount of material stored in a loop must equal the difference between the amount of material input to the loop and the amount output from the loop. One buffer level, alone, cannot be varied because of the loop input/output conditions. If one buffer per loop is ignored, the levels of the remaining buffers can then be specified.

A program was written which implemented the one-buffer-at-a-time version of successive approximations. Process loops were handled by breaking the problem into a number of subproblems, each having a different loop buffer left out of the model. After solving each subproblem, the solutions were averaged, giving added weight to the cheaper solution. The optimization was continued until the grid size was sufficiently reduced by the multi-pass method. An example of how the problem is broken into smaller blocks can be taken from the
model of Figure 1.2. This model can be described by:

\[ x(k+1) = x(k) + Bu(k), \]  

(eqnn. 2-1)

where:

- \( x \) is a vector of buffer levels
- \( u \) is a vector of process rates
- \( B \) is an interconnection matrix describing the model. Element \( b_{ij} \) in \( u \) is the amount of material fed into buffer \( i \) from process \( j \) during one time interval.

Then:

\[ \Delta x(k) = x(k+1) - x(k) = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ -1 & 0 & 0 & 1 & 0 \end{bmatrix} u(k). \]

Note that in order for material flows to be in balance, the sum across a row of \( B \) must be zero. In other words, if all processes are operating at the same rate, there will be no changes in buffer levels. If we define the dependent variables as \( u_d \) and the specified process rates as \( u_f \), then the problem can be restated as:

\[ \Delta x(k) = [B_d | B_f] \begin{bmatrix} u_d(k) \\ u_f(k) \end{bmatrix}, \text{ or} \]

\[ u_d(k) = B_d^{-1} [\Delta x(k) - s(k)], \text{ where} \]

\[ s(k) = B_f u_f(k). \]

Four submodels can be used:

i) \( u(k) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & -1 & -1 & 0 \\ 1 & -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} \Delta x_1(k) \\ \Delta x_2(k) - s(k) \\ \Delta x_3(k) \\ \Delta x_4(k) \end{bmatrix}, \)
\[ u^{(k)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & -1 & -1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x_1(k) \\ \Delta x_2(k) - s(k) \\ \Delta x_3(k) \\ \Delta x_5(k) \end{bmatrix} \]

\[ u^{(k)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x_1(k) \\ \Delta x_2(k) - s(k) \\ \Delta x_4(k) \\ \Delta x_5(k) \end{bmatrix} \]

\[ u^{(k)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x_1(k) \\ \Delta x_3(k) - s(k) \\ \Delta x_4(k) \\ \Delta x_5(k) \end{bmatrix} \]

Care was taken to prevent too rapid a deviation from the trajectory of the previous iteration, so that the subproblem solutions were not so different that the average of these results diverges. The convergence of the cost for these subproblems can be seen in Figure 2.2. The sawtooth appearance of Figure 2.2 is the result of the different solutions obtained from each submodel.

This technique was abandoned because optimizing about one buffer level, with the other buffer levels held at their previous values causes blocks of process rates to vary in unison. Since the cost function (energy usage) is dependent on process rates, the effects of one process's costs will be felt during an iteration involving a buffer distant to that process. The effect of this can be seen by considering a desired shutoff of process P4 of Figure 1.2. For the first
Figure 2.2: Cost Function Value Versus Iteration Number for the "One Buffer at a Time Method"
submodel, processes $P_1$, $P_2$ and $P_3$ would be partially shutoff during the desired shutoff of $P_4$. This occurs because, reducing the level of $T_1$ would require $P_1$, $P_2$, $P_3$ and $P_4$ to reduce their rates if $T_2$, $T_3$ and $T_4$ are to maintain their previous levels. Increasing $T_2$ would result in decreasing $P_2$, $P_3$, $P_4$ and so on. The trajectory which results in decreasing $P_4$ during the desired shutoff period will be selected, since it appears better than others due to the lower rate for $P_4$. This constant bias results in an incorrect solution. Instead of reducing, rates $P_1$ and $P_2$ should increase, to take advantage of the decreased demand resulting from the shutoff of $P_4$.

Considering the direct relation between process rates and energy consumption, optimizing about one process at a time would seem to be a logical approach. Process loops would not create the special problems associated with the one-buffer-at-a-time method. Also, cost function calculations would be simplified when all but one of the processes is not altered. The energy requirements for these fixed processes need only be summed once. This approach was not tried, however, as it would be difficult to ensure that buffer constraints are not violated. Buffer levels are a function of the initial levels and process rates over all time between the initial state and the time in question. A seemingly advantageous process rate may not cause buffer constraints to be violated during the one time period that that process rate is in effect, yet the cumulative effect of process rates chosen in this way may
cause such a violation.

The advantages of both the buffer and process methods were combined in the final computer program by indirectly optimizing with respect to one process at a time. The independent variables are buffer levels, but instead of optimizing with respect to one buffer, the optimization is performed with respect to all buffers connected to the one process. The only buffer levels considered are those that would affect the single process. Memory is required for the quantized levels of only one buffer because of the simple relationship between buffers connected to the same process. An example of this relationship can be seen by considering Figure 1.2. If the level of T1 were reduced by 1 unit from its level of the previous iteration and if only P1 were allowed to change from its previous iteration value, then the level of T2 would be reduced by 1 unit and that of T5 increased by 1 unit. A single vector per process, describing the relative contribution of material to the buffers from this process, provides the necessary information to find the buffer levels. The process rate which would cause these changes in buffer levels is also found easily. For the example above, if 1 unit of buffer level represents 1 hour of 100% production and if a time stage is 3 hours, then the new rate for P1 would be \((1/3) \times (-1) \times 100\% = -33\%\) greater than it was in the previous iteration for this time stage. This information, which is a partial model inverse, is entered as the first element of the buffer level relationship vector just described.
2.4 Unanticipated Problems

Having combined the advantages of both the process and the buffer methods, thus avoiding their respective handicaps, 'second order' problems surfaced. During a process shutoff, some buffer levels will often be taken to their limits in order to accommodate the shutoff and the increased production of the other processes during this time. Again, as an example of this problem, consider the model of Figure 1.2. Assume that P1 is to be shutoff; then, the level of T2 would decrease during this time due to the lack of production from P1. Process P2 should have increased production during the shutoff to smooth total energy demand. This, too, would reduce the level of T2. If T2 were drained to its lower limit by P2 before P1 was shutoff, the shutoff would be blocked. Code was added to the computer program that prevents any buffer level becoming such that shutoffs would not be possible. This is done by looking ahead at processes that are to be shutoff and calculating the minimum/maximum buffer levels in the storages connected to the process, so that a shutoff of the process, at its current rate, would be possible.

Shutoffs, and convergence were held up by effects of quantization. Continuing with the example above, where P1 is to be shutoff; if a time stage were 3 hours and P1 was operating at 1%, then to shutoff P1 the level of T2 would have to be reduced by .03 (units as described above). Such a small change in buffer level may not be available as the buffer
level grid spacing may be too large. A simple addition to the program provided the necessary corrections to the grid points so that complete shutoffs were possible without delay due to too coarse a grid. This correction adjusts the grid points so that in the extreme case (minimum or maximum grid point) the process would be shutoff.

2.5 Conclusions

In its final form, the computer program used for optimizing the mill production schedules is a FORTRAN coding of a successive approximations, multi-pass, dynamic programming algorithm. Complete program listings and flowcharts can be found in Appendix A. The optimization is carried out with respect to one process rate at a time, which is allowed to vary indirectly by altering the levels in buffers that are connected to the process unit. This program is dimensioned for up to 20 processes, 15 buffers, 30 time stages and models having up to 5 buffers connected to a process. With these dimensions, the program requires approximately 12k words of memory. Execution time for an 11 buffer, 12 process model (see Figure 4.2) with 24 time stages is about 10 seconds on the UBC Amdahl 470 V/8. More detailed results are given in Chapter 4.

As well as the dimension restrictions, the mill model must be a constant ratio type. That is, the product of a process must be made up of the same ratio of raw materials,
regardless of the amount or rate of production. Also, every process must be separated from any other process by a buffer, and all buffers must be connected to at least one process. These model configuration restrictions prohibit models that break a large process into smaller interconnected processes, and models that represent large storages as a network of smaller buffers.
3. THE COST FUNCTION AND PROGRAM PARAMETERS

3.1 Introduction

No matter how well an optimization algorithm performs, unless the function being minimized is an accurate representation of the actual costs, the optimization effort is wasted. When only one aspect of a system contributes to the cost function, the task of formulating the performance index is greatly simplified. If the only concern is to minimize the price paid for energy, the cost of steam and the cost of electricity would be needed. More precisely, the actual dollar cost of steam and electricity would not be needed, but the correct relative costs would be required. In practice, optimization of one aspect, exclusively, usually results in poorer performance in regards to other aspects. However, even if one were able to quantify all the salient factors, inclusion of all these factors would require very careful weighting of each to prevent one factor from inappropriately dominating the others.

By examining the problem and reviewing previous work, a list of factors that should be taken into consideration by the optimization program, was produced [HAN, JAN81, CHO80, AAR80, PET69, TIN74, URO80]:

1) energy costs should be minimized,
2) planned pulp/paper production must be met,
production rate changes should be minimized,
planned shutdowns must be scheduled,
random disturbances must be accounted for,
storages must not go empty or overflow,
final buffer levels should be such that the next planning period will not be at a disadvantage,
liquor and chemicals should be in balance,
dynamics (retention times) should be taken into account,
steam may be indirectly stored in black liquor and/or pulp.

3.2 Implementation of the Desired Cost Factors
3.2.1 Factors Handled by Constraints

Constraints can be used to meet some of the above requirements. Minimum and maximum buffer levels, final buffer levels and planned pulp/paper production can all be specified as problem constraints. Buffer level constraints are chosen by taking into account the mean time to repair adjacent process units. Buffer levels are constrained so that there is enough material in reserve to allow 100% production for the mean time to repair (MTTR) a preceding process. Sufficient 'head room' is also maintained so that 100% production can continue during the MTTR for a process on the output side of a buffer. Using these statistics to set buffer levels has been used in JAN81, URO80, PET69, TIN74. For use in this project, statistics were
obtained from MacMillan Bloedel Limited in Vancouver. These statistics are simply the mean time between failures (MTBF) and the MTTR for each production unit of the company's pulp and paper mill at Powell River. The MTBF measure is used to specify the maximum recovery time of buffer levels after a disturbance. That is, after using the reserve buffer capacities to smooth out the effects of a breakdown, these capacities must be recharged within the MTBF.

Including random disturbances directly in the cost function, results in a more elaborate recursive dynamic programming relation [SHO81]:

\[
J(k,x) = \min_{u(k)} \{ S(x,u,k) + \sum_{w} P[x(k+1)=w|x,u,k] \cdot J(k+1,w) \} \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \q
3.2.2 Material Balances and Process Dynamics

Material balances and process dynamics are not necessary due to the simplicity of the mill model used. A fixed, linear relationship between process input requirements and output rate is assumed, so that an overall, approximate, material balance is obtained by summing the flows into buffers. Details of the consumption and production of the different chemicals and materials required for each process are ignored. Leaving out process dynamics and thereby assuming complete, instantaneous response to desired rate changes, allows the use of a simple first order model as given in equation 2-1. Since the planning time stages are long (2 to 8 hours) relative to the time delays of the processes, the use of this simplified model has been found to be acceptable. Leiviska and Uronen used an 8 hour (1 shift) time stage with both the Tamura and "out-of-kilter" algorithms [URO80]. Pettersson used a 6 hour period [PET69] and Tinnis used a time step from 1 to 8 hours with linear programming [TIN74].

3.2.3 Bottleneck Departments

Special consideration in the cost function to "bottleneck" departments should not be required as the mill model and buffer level constraints force such units to be taken into account automatically. The 'gross' material balance specified by the mill model forces all departments to produce
at a rate sufficient to meet the desired pulp/paper production. If the "bottleneck" department is a problem because of frequent breakdowns, then the buffer level constraints would reflect this and require significant reserve production from this process.

3.2.4 Planned Shutoffs

Planned shutoffs must be scheduled so that maintenance and similar interruptions can be accommodated without causing undue disruption to the rest of the mill. Shutoffs introduced at the beginning of the optimization period represent unexpected breakdowns that have caused a new optimization run. Two methods of introducing these shutoffs were considered: process rate constraints and penalty factors. A shutoff could be obtained by insisting, by means of constraints, that the process rate be zero during the desired time. Unfortunately, the limited grid of the multi-pass method may not provide the possibility of a shutoff. If 'stiff' constraints are used (i.e., when constraint violations are not allowed) an acceptable trajectory may not be found because of insufficient grid point resolution or range. 'Soft' constraints, which allow violations, but discourage them, can be implemented by a penalty in the cost function. The penalty approach is the method used with this project.

Each process, at each time stage, is assigned a penalty factor. The square root of the process rate is multiplied by
this penalty factor and added to the total cost. The square root of the process rate is used so a more complete shutoff can be obtained. Small values of process rates result in a small contribution to the total cost. A desired shutoff may be only partially obtained when the associated penalty loses its significance in the total cost. By using the square root of process rates, much smaller rates can have significance in the total cost.

3.2.5 Process Rate Changes

When a change is made in the operating point of a process unit, product degradation often results from the transients produced by the change, and the time required to make fine adjustments to the process in order to bring it to the new rate. Avoiding these changes sometimes avoids product degradation and so, reduces waste and energy requirements, and increases profits. Also, an optimization algorithm, as with all methods for introducing tighter control of a process, naturally tends to a situation in which variables internal to the system are adjusted rapidly in order to obtain smooth trajectories in the external variables. It is thus normal to penalize too rapid adjustment of the internal control quantities. A penalty for rate changes is included in the cost function. This penalty is used in two modes, a normal and a shutoff mode. During shutoffs and other disturbances the penalty is greatly reduced because if a rate-change penalty
was in effect during the start of a shutoff, it would be in conflict with the shutoff penalty. The shutoff penalty would encourage a rate change from a high, pre-shutoff, rate to zero, while the rate-change penalty would discourage such a drastic change.

3.3 Energy System Simplifications
3.3.1 Boiler Load Allocation

The primary concern is to reduce energy costs which are therefore the major component in the cost function. A mill may operate a number of power boilers, recovery boilers, turbogenerators, and have a connection into a utility grid and in some instances, even hydro-electric dams. The interconnection of these units can be quite intricate from an energy point of view. Electrical energy is used heavily in some areas, for example paper machine drives and mechanical pulp grinders, and electrical power is also generated internally using back pressure turbogenerators. High pressure boilers generate steam within the mill by burning process by-products such as hog fuel or black liquor from the kraft recovery loop. This steam is then reduced to process pressures by expansion through the turbogenerators and pressure reducing valves. Both high pressure process steam supplies, commonly for digesters, and low pressure supplies, for evaporators, paper machine driers and other applications, must be kept in step with demand at all times. Since turbogenerator settings
are dependent on the demand for high and low pressure steam, the steam and electrical supplies interact.

For every combination of steam and electrical demand, there is a combination of the steam and electrical generation settings that meets the energy requirements at minimum cost. Trade-offs are possible between the various boilers, and between the steam and the electrical systems. Purchased electrical power is normally reduced by using process steam to produce electricity through turbogenerators before use in the mill. The use of expensive fossil fuels can be reduced by using more steam from boilers that burn hog fuel, sawdust and other pulping wastes. Overall boiler efficiency is a function of age, condition, and load. Generally the efficiency decreases with increased load. If the steam generation capacity was unlimited, the best combination of boilers would be that which results in the incremental costs of steam production being equal for all boilers [HAN]. The first example of the gradient method, given in Chapter 2, verifies this for the case of two boilers with costs that depend on the square of the load. To say that the incremental costs should be equal is equivalent to saying each additional unit of energy should be produced by the cheapest source. A boiler load optimization method that utilizes this unit load assignment approach has been implemented by Leffler [LEF78]. In the constrained case, the best operating point is where incremental costs are equal for all boilers not at their limit, and those boilers that never reach an incremental cost
3.3.2 Mill Steam Systems

Static boiler load optimization has been considered by a number of researchers [HAN, LEF78, HUN80, CHO80]. This previous work considers the problem of matching supply and demand for both high and low pressure process steam as well as that of the overall steam production. The steam supply system of a pulp and paper mill usually consists of boilers producing high pressure steam (greater than 600 psi), pressure reducing turbogenerators, condensing turbogenerators, pressure reducing valves and desuperheaters. Besides the high pressure steam, two lower pressures are used, usually at pressures of about 165 psi and 60 psi (see Figure 1.1). These lower pressures are obtained from the exhaust and extraction stages of pressure reducing turbines, or from pressure reducing valves. Water is added in desuperheaters to reduce the steam temperature to the desired level of saturation. Thus, when all three steam pressures are considered, the steam allocation problem is complicated by the variety of ways of satisfying the low pressure demand. Efficiencies of the pressure reducing methods are similar (96 to 98%) and so for most operating conditions, the total high pressure steam demand is a sum of the steam demands of each process, weighted to account for pressure reductions.
3.3.3 Turbogenerators

Previous work on the static optimization of mill energy systems considered the nonlinear relationships between the extraction flows, condensing flows and electrical power production of turbogenerators. Nonlinear optimization methods [CHO80] and linearized turbogenerator models [HUN80] have been used to determine the best flows for given steam demands. Although the relations between steam flow and electrical power are very nonlinear, the total efficiency of turbogenerators, over their normal operating range, is quite constant [NEW79]. A power balance, instead of a steam balance, avoids the nonlinear complications of the turbogenerator and simplifies the procedure of finding the total steam load. If the above simplification is used, a one-to-one relationship between total steam power demand (including electrical power produced by turbogenerators) and the cost of meeting this demand exists. The actual setting of pressure reducing valves and turbogenerator extractions would be determined from the demands for lower pressure steam and would have little effect on the total steam power demand. The load assignment to the boilers could be found by the incremental cost method.
3.3.4 Grouping Power Sources

The objective of this thesis is to reduce energy costs by scheduling. Methods of best meeting an energy demand are well developed; hence, the power balance simplification and power source grouping are used. All power boilers are grouped as one large boiler having a composite cost-load relation. All turbogenerators are lumped as one with an aggregate efficiency. Recovery boilers are not included in the power boiler grouping since their fuel is the black liquor of the kraft recovery process and so are part of a material feedback loop in the mill model itself. Individual recovery boiler rates directly affect buffer levels; therefore, they influence the production schedule of the remainder of the mill. A recovery boiler can be viewed as a means of obtaining steam that has been 'stored' in black liquor. Recovery of the 'stored' steam is an integral part of energy management, and so recovery boilers are not included with the power boiler group. A disadvantage of grouping energy sources is that variations of availability within a grouping require a new composite performance characteristic to be determined.

Composite boiler costs could be easily found from the individual boiler characteristics and the incremental load assignment method of Leffler [LEF78]. As each increment of load is assigned, the resulting boiler allocation and total cost can be tabulated. For use with the optimization program, the composite load-cost relationship is approximated by a
polynomial.

Once the steam load is determined from the schedule optimization program, individual boiler loads can be found from the table generated during the construction of the composite cost curve.

3.3.5 The Simplified Energy System

The simplified energy system consists of: lumped power boilers, lumped turbogenerators, a connection into an electrical utility, and process units that have steam and electrical demands depending on the process rates. The various steam pressure demands are combined into a total steam power demand for each process. Recovery boilers are represented as having a negative steam demand, and hydro-electric dams imply negative electrical demands.

The dynamic programming algorithm finds the best schedule by comparing the costs of different operating rates. Using the simplified model, energy costs are found by summing the steam and electrical demands of all processes, then finding the best combination of steam production, turbogenerator rate and purchased power rate. This combination is found by the incremental cost criterion explained above. The composite power boiler performance is represented by a polynomial of degree 3 which approximates the load-cost relationship. Purchased power billing rates are represented by
a two-slope, time varying, piece-wise linear function. Finding the best operating point, from the total power demands and the cost relationships can be accomplished with a short series of FORTRAN "IF" statements (see Figure 3.1). The cost calculation is repeated many times in the dynamic programming algorithm. Reducing the complexity of this calculation has a significant effect on the overall execution time. If the simplifications used here cannot be made for a certain mill, the generality of dynamic programming allows very detailed cost calculations to be made, at the expense of added execution time.

3.4 Parameter Selection

In this section, the selection of program parameters is discussed. The parameters concerned arise mainly from the multi-pass and successive approximation methods. The multi-pass method affords a reduction in the number of state (buffer level) grid points. How many grid points should be used? How quickly should the grid size be reduced to the final 'fine' grid? Successive approximations reduces computation and memory requirements by repeated optimization with respect to a subset of the problem variables. How many iterations should be used so that the successive approximation method converges? Another parameter, one that must be chosen in all dynamic programming applications, is the length of the time step.
TEM*EFF is the maximum electrical power demand that does not have a demand charge.

EFF is the turbogenerator efficiency.

TG = 0
EL = total mill electrical demand
STM = total mill steam demand, with TG = 0

EL > TEM * EFF
and
STM < TSM2

YES

A = min(TGM, EL/EFF - TEM, TSM2 - STM)
TG = STM - A
EL = EL - EFF * A

NO

STM < TSM1

YES

A = min(TGM - TG, TSM1 - STM, EL/EFF)
TG = STM + A
EL = EL - A * EFF

NO

TGM is the maximum turbogenerator electrical output.

TG = turbogenerator steam used to produce electricity
EL = purchased electrical power
STM = steam production

Find cost(EL, STM)

Figure 3.1: Steam Production, Turbogenerator Rate and Purchased Power Assignment Logic
3.4.1 Number of Grid Points

In terms of the number of computations required, the fewer grid points the better. Appendix B gives a short analysis of the amount of computation required for different numbers of grid points assuming equal state space coverage and equal final grid point spacing. The allowed buffer levels of each pass of the multi-pass method are a subset of the total state space (physical buffer level limits). If on each iteration the trajectory is changed in the same direction by the maximum change allowed by the current subset, the total change will be described as the 'state space coverage'. An odd number of points has the advantage of allowing the trajectory of the previous iteration to set the middle grid points of the next. By ensuring that the previous trajectory is a possible 'new' trajectory, a trajectory better than, or at least as good as, the previous trajectory can always be found. If an even number of grid points were centred around the trajectory, no grid point would correspond to this trajectory. If the 'old' trajectory is not included as a possible 'new' trajectory, there is the possibility of not finding a 'new' trajectory because the initially coarse grid of the multi-pass method may not include a feasible solution in some extreme cases.

Trials with 3, 5, and 7 grid points per time stage were made. 'Coverage' of the state space and final grid point spacing were equal for the comparisons. The computational
requirements predicted in Appendix B were confirmed and only slight differences in the solutions were observed for simple production schedules. For more difficult cases that included shutoffs, the 3 point grid produced inferior results. Convergence of the 3 point case to the solution obtained with the 5 and 7 point versions required more than 3 times the predicted number of iterations in some cases. The initial trajectories were the same for the comparisons. This initial trajectory is not near the solution in cases that include shutoffs, and so, there is a possibility that the 3 grid point method does not have as good a capability to produce such a large change in the trajectories as the 5 and 7 grid point methods. Although it is suspected that the smaller range of buffer levels considered on each pass with the 3 point grid, coupled with the successive approximation method is responsible for the very slow convergence, the exact cause of this problem was not investigated. The solutions found with the 7 point method were no better than those found with the 5 point method, even when the schedules included shutoffs.

3.4.2 Number of Iterations

Since the successive approximations method may require a number of iterations to reach convergence, the rate at which constraints are 'closed in' around the trajectory must be chosen such that convergence is not hampered. This is arranged by performing a number of iterations with a fixed grid spacing before reducing the grid spacing. Previous work with
successive approximations, applied to a water reservoir system [LAR70], and a power generator allocation problem [ROS80], found that convergence is obtained usually within only 3 iterations. Results with 3 iterations per grid point spacing provided quick, monotonic convergence. Trials with as many as 8 iterations per grid point spacing resulted in excessive computation time and no improvement in the solution.

The rate selected for constraint 'closure' and hence grid point spacing reduction, is based on the desired initial grid size, final grid size, and 'state space coverage'. Too rapid a reduction of grid size or too few iterations will not provide complete 'state space coverage'. The initial grid size provides the largest single contribution to 'state space coverage'. Choosing a large initial grid spacing allows rapid changes in the trajectory. Using very large initial grid spacing is not advisable, however, because such large changes in the trajectory may not be possible, due to process rate limits, and a rapid grid size reduction would be required if the final grid size is to be obtained within a reasonable number of iterations. An upper limit on the initial grid spacing can be calculated from the change in buffer level produced by a maximum process rate. Too rapid a reduction in grid size may result in the final solution being constrained to being close to the initial 'rough' solution. Trials with different rates of constraint 'closure' showed that a 50% reduction in permissible buffer levels, on each iteration, allowed convergence to the optimum. When the constraints were
reduced to 20% of their previous range, shutoffs were not obtained and the production schedule called for unnecessary rate changes. The final grid spacing is chosen such that the spacing is less than the measurement and control accuracy of the mill.

3.4.3 Iteration Order

We are free to choose the order in which processes are adjusted toward the optimum. Tests with different iteration orders did not have significantly different solutions or faster convergence with any of the four models used. Slight improvements were observed when the first process to be selected was the only process to be shut off. This slight advantage is insignificant after about 4 iterations.

3.4.4 The Parameters Used

Successful program performance is obtained when the constraints are reduced to about 80% of their previous range after each set of 4 iterations with a fixed grid spacing of 5 grid points. Although experience showed that a 50% reduction gave good results the smaller, 20%, reduction is used to ensure that the solution is not restricted to be too close to the initial coarse grid solution. The slower constraint 'closure' also ensures complete 'state space coverage'. The 4 iterations per grid spacing was selected on the basis of other researcher's experience [LAR70, ROS80], and on the results
obtained with this number of iterations (see section 3.4.2). Initial grid spacing is set to less than the smallest of the maximum fractional change in tank levels possible with 100% process rates. For example, in the 5 buffer model of Figure 1.2, the largest difference between maximum and minimum buffer levels is 12 hours of 100% production. A planning time stage is 3 hours; therefore, 3/12 or .25 of the full constraint range is used as the maximum starting constraint range. In practice there are few instances when this maximum level change would be required (i.e., shutoffs) and so the starting grid is made smaller than this maximum. For the example above, 0.2 was used, giving a 10%, of the physical buffer limits, spacing between the grid points. With these parameters, a total of 10 multi-pass iterations are required to reach a final grid size of at most 1/75 of the complete buffer level range.

Often optimization algorithms incorporate a "stopping criterion". When little or no improvement is achieved by the previous few iterations, convergence is assumed to have been reached and the procedure is halted. When the initial and final grid size is predetermined and the rate at which the grid spacing can be reduced is specified, stopping criteria are not needed and are not used.

Finally, the length of the planning time stage must be specified. Since the mill is modelled as a simple first order system, the time stage need be no shorter than the shortest
production run. Previous researchers have used a 1 to 8 hour time step and a total planning period of 2 to 3 days [PET69, TIN74, JAN81, URO80]. Most of the program tests were done with a 3 hour time step with 1 hour time stages used for a few tests. The shorter time step required more iterations because the initial grid size must be smaller than with the 3 hour time step case. A smaller initial grid spacing reduces 'state space (buffer level) coverage' which must be countered with increased iterations.
4. RESULTS

The optimization program was used to produce schedules for a number of mill models, under a variety of circumstances. The simplest model used was a 4 process mill, configured in a straight line (process i fed process i+1 through tank i). This small model was useful for program debugging and verification. Most of the program runs were done with the 5 process model of Figure 1.2 because although this model is small, it contains a recovery loop. In order to check program performance with models containing multiple feedback loops, cases were run with the model of Figure 4.1. The most complete model used is shown in Figure 4.2. This 12 process model represents both the groundwood and kraft pulping sections of a mill.

4.1 Process Description

A general understanding of the process steps in an integrated pulp and paper mill will make the results of the optimization more understandable. Referring to Figure 4.2, the incoming wood supply (processes P1 and P2) passes through a series of stages before emerging as paper or market pulp (processes P10, P11, P12).

Wood pulp is the fibre state intermediate between wood
Figure 4.1: A Multiple Loop, 8 Process Mill Model

Key:  Tx buffers,  
Px processes.

This is a fictitious mill.  
The function of the processes are not specified.
Figure 4.2: A Twelve Process Mill Model

Key:
P1 - groundwood
P2 - sawmill
P3 - refiner
P4 - digester
P5 - screens
P6 - cleaners
P7 - evaporator
P8 - recovery boiler
P9 - kiln
P10, P12 - paper machines
P11 - pulp machine
T1 - mechanical pulp
T2 - sawdust storage
T3 - wood chip silos
T4, T5 - blow tanks
T6, T7 - kraft pulp
T8 - weak black liquor
T9 - strong black liquor
T10 - green liquor
T11 - white liquor
and paper. In the mills considered here, pulp is produced from wood either mechanically or chemically. The abrasive techniques used to produce pulp mechanically usually have the disadvantage of reducing the fibre length and so weakening the strength of paper made from such pulp. In the chemical, or kraft process, the fibre lengths are preserved by chemically dissolving the lignin in a strong alkaline solution; kraft pulp thus makes strong paper. The chemical process, however, has the disadvantage of leaving the pulp discoloured, and several stages of bleaching are necessary if the end product is to avoid the brown shade of a grocery bag. Most paper is produced from a skillfully blended combination of mechanical and kraft pulp. The mix is determined not only by the process, but also by factors such as the species of the tree, and its quality.

The way in which the pulp is produced has a substantial impact on the energy usage within a mill. Mechanical pulping is energy intensive. In the stone groundwood process, blocks of wood are forced against rotating grind stones with the mechanical energy usually furnished by large synchronous motors, although water turbines are also used. This process is a heavy user of electrical energy and it is by manipulating the grinder load that mills have traditionally controlled overall electrical demand. Clearly, the synchronous motors are also available for power factor correction. More recently the thermo-mechanical pulping process has become widespread. Here, the wood is processed in the form of chips, which are forced,
in steam, between counter rotating refiner blades. Again the motive power is electrical and synchronous motors are used. Thermo-mechanical pulp has more strength than stone groundwood and its introduction often reduces the proportion of kraft pulp needed to produce paper of a given quality.

The kraft process uses chemical energy for the pulping and, although this reduces the overall utility demand enormously, the difficulties of recycling the chemicals and maintaining pollution control standards are substantial. Although the net absorption of energy in the kraft cycle is low, different stages of the cycle are large generators or absorbers of energy. Wood chips and the alkaline "white liquor" are cooked under pressure in a digester, either as a batch operation or continuously. This stage absorbs substantial quantities of steam energy. After digesting, pulp and the spent "black liquor" carrying the lignin are separated and the pulp passes on to bleaching. Black liquor contains combustible organic solids and is concentrated by passing it through evaporators - a multiple stage heating process which again absorbs copious amounts of steam. The "strong black liquor" emerging from the evaporators is then used as fuel in the recovery boiler which generates energy roughly equal to that expended elsewhere in the loop. In the final stage, the spent combustion products are recovered and recasticized in the lime kiln. A combination of kraft and mechanical pulp is used in the paper machines. Paper machines are complex electro-mechanical systems that require substantial electrical
energy for drives, and low pressure steam for drying. In some cases, a pulp drier is used to produce dry kraft pulp that is to be broken down again and used elsewhere.

Table 4.1 shows roughly the energy generation and usage of the different components of a typical mill. It is evident that utility management for such an involved process is highly complex, especially since the bulk of the materials being handled normally prevents any large scale storage between process blocks.

4.2 Program Verification

Verification of the program was carried out using it to solve small, simple problems that can be calculated by hand or intuitively. An example that can be solved intuitively is one in which the initial and final tank levels are equal, output production rates are constant and all other aspects of the mill are constant (i.e., no shutoffs, no billing rate changes for purchased power). The solution for this case is a constant production rate for all processes, equal to the output production rate. This is the solution found by the computer program. Another case that has an intuitive solution is the same problem stated above, except with a range of buffer levels allowed at the final planning time. If no process produces energy, the solution should have constant process rates as before, but this time the rates should be such that all final tank levels are at the minimum allowed level. The
### Table 4.1: Energy Generation and Usage in a Typical Pulp and Paper Mill

<table>
<thead>
<tr>
<th>Energy used (GJ/t)</th>
<th>Steam</th>
<th>Electrical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kraft mill:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>total steam demand</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>total electrical demand</td>
<td></td>
<td>3.1</td>
</tr>
<tr>
<td>1. digester</td>
<td>3.5</td>
<td></td>
</tr>
<tr>
<td>2. washing &amp; screening</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>3. bleach plant</td>
<td>3.5</td>
<td></td>
</tr>
<tr>
<td>4. drier</td>
<td>4.4</td>
<td></td>
</tr>
<tr>
<td>5. evaporators</td>
<td>4.1</td>
<td></td>
</tr>
<tr>
<td>6. boiler auxiliaries</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>7. turbogenerator</td>
<td>1.9</td>
<td></td>
</tr>
<tr>
<td>8. miscellaneous</td>
<td>4.6</td>
<td></td>
</tr>
<tr>
<td>Thermo-mechanical pulping</td>
<td></td>
<td>5.1</td>
</tr>
<tr>
<td>Groundwood pulping</td>
<td></td>
<td>6.7</td>
</tr>
<tr>
<td>Paper machine drives</td>
<td></td>
<td>3.8</td>
</tr>
</tbody>
</table>

**Energy supplies**

<table>
<thead>
<tr>
<th>Recovery boilers</th>
<th>55%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hog fuel</td>
<td>29%</td>
</tr>
<tr>
<td>Fossil fuel</td>
<td>16%</td>
</tr>
<tr>
<td>Turbogenerator</td>
<td>25%</td>
</tr>
<tr>
<td>Purchased power</td>
<td>75%</td>
</tr>
</tbody>
</table>

[in part, from RES81]
A computer program produces this solution for models not having feedback or feedforward loops. When a loop is included in the model, specific loop buffer levels are not attainable because such systems are uncontrollable. All buffers cannot be taken to their lower limits because such a system state may require a process which directly or indirectly both feeds and drains the loop, to have two different rates, one for feeding and one for draining. This is not allowed, because it implies that the process has the capability to store material. The constant output production test, when applied to a model containing a loop, produced a constant rate schedule that ended with the first buffer in the loop filled to its upper limit and the last buffer in the loop emptied to its lower limit. These two buffer levels indicate that the loop as a whole is operating at the minimum rate required to meet the desired output production.

Another set of tests was carried out in which the energy cost characteristics were constant over the entire planning period. Under these circumstances the optimal schedule is one in which the steam demand is smoothed over time, and the purchase of peak power is minimized. Figure 4.3 shows results obtained for the model of Figure 4.1 when the output production rate changes frequently. Buffer level units are in hours of 100% production and process rates are normalized to 1 for 100% production. The dashed lines plotted with the buffer levels are the buffer level constraints. Processes P1 to P7 have slight rate variations in order to
Figure 4.3: Erratic Output Production
(For the Model of Figure 4.1)
Figure 4.3 continued.
compensate for the swings in P8. Note that total steam demand is practically constant, and no peak power is purchased. The optimization procedure has provided a schedule that allows smooth operation of the mill, while almost completely masking production variations from the energy sources.

For the case shown in Figure 4.4, process P2 is shut off for 2 time stages (6 hours) yet the steam demand stays constant over the whole planning time range. Process rates for P1 and P4 are increased during the shutdown of P2 in order to take advantage of the energy demand reduction that occurs from the shutdown. By increasing production during the shutdown, processes P1 and P4 may produce less during the other times, thereby smoothing the overall energy demand. At the same time as providing a constant steam demand, the optimal schedule does not require peak power to be purchased at any time.

A breakdown of the digester (P1) is demonstrated in Figure 4.5. Process P4 has an increased rate during the breakdown to compensate for the reduced energy consumption of this period. The recovery boiler (P3) is practically shut off in order to smooth the steam demand by saving fuel for future, higher demand, periods. The evaporator (P2) cannot aid P4 in compensating for the digester breakdown as the weak black liquor storage tank (T2) limits P2 to a rate no greater than that which empties T2 to the lower constraint level.

A more elaborate mill model shown in Figure 4.2 was used with the optimization routine to produce the results of
Figure 4.4: Planned Shutoff of P2 During Times 6, 7
(For the Model of Figure 1.2)
Figure 4.4 continued:
Figure 4.5: Breakdown of P1 During Times 1, 2
(For the Model of Figure 1.2)

continued...
Figure 4.5 continued:
Figure 4.6. In this case, a breakdown of the kiln (P9) is followed by a planned shutdown of the refiner (P3). Again, the total steam demand is smoothed and there is no peak power purchased.

When the power source cost characteristics change during the planning period, a smooth steam demand can no longer be expected. For example, Figure 4.7 shows the optimal schedule for a case where electrical power is less expensive during the first 6 planning periods. Although more electrical power is purchased during the cheaper periods than the last 6 periods, the steam production is the opposite. The steam production takes a jump up at the seventh time period, as does the turbogenerator load, in order to produce more electricity within the mill by the turbogenerator. Had a constant process rate, constant steam demand schedule been used in this case, the energy costs would have been 5% greater than for the solution of Figure 4.7.

4.3 Savings Obtained

The savings obtained by using the program depends on the schedule that would have been used had the optimization not been carried out, on the steam and electricity costs, and on the buffer capacities. The production schedule used as a starting point for the optimization program is a smoothed version of the output schedule. Such a schedule is reasonable for cases that do not include shutoffs. If is is assumed that
Figure 4.6: Breakdown of P9 During Times 1, 2 and A Planned Shutoff of P3 During Times 8, 9 (For the Model of Figure 4.2)
Figure 4.6 continued:
Figure 4.7: Purchased Power Billing Rate Increase at Time 7
(For the Model of Figure 1.2)
Figure 4.7 continued:

[Graph showing a time series with labeled axes]
the best steam production - purchased power combination is used in all cases, then for the example of Figure 4.7, the optimum schedule has a 5% cost reduction over the initial schedule. A large, 50%, saving was obtained in the case shown in Figure 4.3 because a peak charge of 3 times the base rate makes purchases of peak power very costly.

Not only are similar savings possible in cases that include shutoffs, but the program's ability to schedule a process shutoff is useful in its own right. For the case shown in Figure 4.4, the shutoff of P2 is scheduled and the effect of this disturbance is almost completely compensated for by the remaining processes. The resulting energy cost is the same as the equivalent schedule without a shutoff.

4.4 Impossible Schedules

Circumstances which will cause the optimization program to fail can be classed as either initially feasible or initially infeasible. An initially infeasible case is detected by the subroutine that finds the initial schedule. If a starting schedule cannot be found that does not violate constraints, then the schedule initialization routine halts the program. This problem could arise in the case of a plan to operate the output production units at greater than 100% for a long duration. The remainder of the mill would not be able to keep buffers from running dry, and so this case would be declared initially infeasible and execution halted.
Initially feasible cases can be initialized, but then a shutoff (not part of the initialization) cannot be achieved. In these cases, a schedule with a maximum reduction in production rate, instead of a shutoff, is produced by the optimization program. A situation in which such an incomplete shutoff would result could be, for example, if we wish to shut off P1 of Figure 1.2 for 3 time stages, while maintaining 100% production of P5. The capacity of buffer T1 would not permit this.

4.5 Conclusions

A successive approximations, multi-pass dynamic programming algorithm can be used to optimize the production schedules of mills such as pulp and paper mills. Process shutdowns, process rate and buffer level constraints, and specified production demands are accounted for. This optimization program can be used as an aid to plan production schedules in order to minimize energy costs and reduce the effects of disturbances in the mill.
The effectiveness of the production scheduling program could be lost if it were inconvenient to use. The program is intended for use by mill operation superintendents who usually are not familiar with programming or the use of computers. In the event of a disruption within the mill, a superintendent may wish to have results from a scheduling aid in very short notice. There must be an interface between this group of users and the optimization program that enables simple and rapid schedule optimization. If the program is awkward to set up, if results are difficult to interpret, or if the method of program operation is prone to errors, then its use may be abandoned in favour of less effective but more convenient methods. Computer graphics can be used to make the program quick and easy to use.

5.1 Graphical Input

Creation of an input data file, by a keyboard, for the optimization program is a tedious, time consuming and error prone task. A 15 buffer, 20 process, 30 time stage model having 2 process loops would require an input file of approximately 1300 numerical entries. Fortunately, accuracy of about 1% is more than adequate for most entries because of the limited accuracy of mill measurements and controls, and so
A colour graphics program for data input was written in FORTRAN and LIG [SCH78]. Values are entered using the graphics cursor to 'draw' the desired levels on a displayed grid of magnitude versus time. These values are then written into the input data file. The interactive graphics language LIG is used because of the simple (in terms of programming) way in which it provides a colour graphics extension to FORTRAN. Graphics statements are placed in the source code with FORTRAN statements as if LIG and FORTRAN were one language. A LIG "preprocessor" replaces the LIG statements with the appropriate FORTRAN subroutine calls before compilation. Another reason for using LIG is its "graphical variable" type. The displays needed for input/output are repeated many times during a scheduling session (e.g., grid lines, schematic shapes and arrow heads). LIG allows a graphical variable to be defined as a certain image and then to display this, or a similar image, at a later time; the graphical variable is used to refer to the entire predefined image.

The input program is used to graphically enter values that commonly change from one planning time stage to the next, such as buffer level constraints, output production schedules, penalty factors and turbogenerator limits. Purchased power billing rates and initial buffer levels are also entered using the input program except these entries are made using the
terminal keyboard. Values of the mill model matrices and related information, for example energy consumption rates, are not altered by the data input program.

Figure 5.1 is a photograph of the computer terminal screen, taken during the input of the minimum and maximum allowable tank levels. The tank level constraints are shown in red, yellow and blue on a green grid of magnitude versus time. Physically impossible values are prevented by restricting the range of values represented by the green grid. After a buffer or process is selected, the current level (or rate) is displayed in red. As the desired changes to the values are made, the new values are displayed in yellow. Once all changes have been made, the new trajectory is displayed in blue in order to confirm the input, and thus preventing errors.

Data are input by positioning the cursor to one end of a line segment representing data values and then pressing the 'B' (for begin entry) key. The coordinates of this point are displayed in the upper right hand corner of the screen to confirm the cursor position. The cursor is then moved to the other end of the desired line segment and 'E' (for end entry) is depressed. The indicated line segment is then displayed in yellow. Pressing the 'V' (for verify) key during the session causes the cursor position to be printed in the upper right hand corner of the screen. Once all changes to the parameter in question have been made, 'S' (for stop) is pressed to request the display of the blue confirmation curve and to
Figure 5.1: Photograph of Graphics Terminal Display During the Input of Buffer Level Constraints.

Figure 5.2: Colour Display of Optimal Trajectories Within a Small Mill Schematic. (Photograph of Graphics Terminal Display)
allow a different parameter to be selected for input.

Having to create an entire input file for each schedule optimization is avoided by using a 'normal' file as a starting point for data input. This 'normal' file contains average values for output production with no process shutoffs. This file is copied into a 'working' file which is then modified with the graphical input program. In this way, only a few entries in the working file need be changed in most cases. The optimization program does not affect the input file; hence, if the resulting schedule is unreasonably restrained by a buffer level constraint or if an impossible production demand was requested, the minor changes required to the input data can be made quickly using the input program without restarting the input process from the 'normal' file. Program listings and flow charts for the graphical input program are presented in Appendix C.

5.2 Graphical Output

After entering the desired data input file, the optimization program is run and process rate, buffer level, and energy demand trajectories are produced. A colour graphics display is used to present these trajectories as graphs placed appropriately within a schematic of the mill model. Figure 5.2 is a photograph of a display for a 5 process model. The trajectories in the photograph are the solution to a scheduling problem that includes a breakdown of the digester
during the first 2 time periods, as in Figure 4.5. The validity of the trajectories can be quickly verified from the mill schematic, buffer level and process rate display. More detailed plots of buffer levels, process rates, and energy demands can be selected for display so that levels, rates and the action of constraints can be checked. The alternative to viewing these displays, looking at lists of printed numbers, is a confusing and tiring option. For program listings and flow charts of this output program, see Appendix D.

Using computer graphics with the production scheduling optimization program proved to be most convenient. Data entry is simple and rapid. Graphically displaying results gave a quick feedback of the effects of the production demands, process shutoffs and buffer level constraints.
6. DISCUSSION

6.1 Summary

Pulp and paper manufacture is an energy intensive process. The different process units of a mill require different amounts of energy and different ratios of steam to electrical energy. Energy is supplied to the mill by power boilers, recovery boilers (which are a stage of the chemical pulping process), hydro-electric dams, and electrical utilities. Turbogenerators provide in-mill electrical generation, thus allowing increased steam production to be traded for a reduced demand for power from the utility.

Savings in the price paid for energy are possible by scheduling the process units of the mill in such a way that demand charges for purchased power and inefficient boiler operating points are avoided. This schedule is achieved by effectively using the capacities of buffers to relax the interdependence of process rates.

Bellman's dynamic programming algorithm was selected for use to find the optimum production schedules because the simplicity of the algorithm results in straightforward program code, it does not need derivatives of cost or constraint functions, time dependent constraints are handled easily, and very general cost functions may be used. Both the successive
approximations and the multi-pass modifications were applied to the algorithm in order to overcome the "curse of dimensionality".

The program was used to find optimum production schedules for mills that were modelled as production units, interconnected by storage units. It was assumed that any delays in the response of process units are negligible in comparison to the decision time step of at least one hour. Energy costs were reduced by scheduling in such a way that the steam demand was smoothed and peak power demands were avoided. Process rate changes were kept to a minimum, so that product degradation would not result from the production plan, and process shutdowns were scheduled for the times requested. Interactive colour graphics were used to make the program easy and fast enough to be suitable for use as a planning aid in a pulp and paper mill environment.

6.2 Future Work

Having a program that is capable of optimizing the production schedules, the next step is to verify its performance in a real mill. Investigations would have to be made into the accuracy of the mill model and the cost function. Perhaps a more detailed energy demand calculation, or a more elaborate mill model would be needed? During the trial, statistics on the mill performance could be collected for use in setting buffer level constraints and for accessing
the savings made possible with the program.

After making any necessary modifications to the program, an attractive possibility is to include the production optimization program in a mill-wide monitoring and control supervisory computer. Such an arrangement could automatically provide information, to the program, on buffer levels, process rates, and energy consumption. With this system, the results of the schedule optimization could be used to set the operating targets of the process unit controllers. The monitoring aspect of the supervisory computer would provide mill superintendents with a real-time mill status display that would alert them to disturbances and verify that the optimal schedule is being followed.

Use of a supervisory computer requires mill integration. In order to integrate a mill, much work would have to be done on mill computer communications and protocols because of the multi-processor, hierarchial structure of mill monitoring and control [HAG81]. Integration would also require work in the area of performance and maintenance of sensors. Presently, the level of many storage tanks and silos is measured crudely if at all.
6.3 Conclusions

In this thesis, it is shown that a successive approximations, multi-pass variation of Bellman's dynamic programming can be used to reduce energy costs in pulp and paper mills. Cost reductions are obtained by finding production schedules that minimize the need to purchase peak electrical power and smooth the total steam power demand.

Inclusion, in the cost function, of the appropriate penalties is an effective method of scheduling process shutoffs and breakdowns. Penalty factors are also used to ensure that the production schedules produced by the program do not contain unnecessary rate changes that would affect product quality. Because of the generality of dynamic programming, no problems are encountered in accounting for the many constraints imposed by buffer level, process rate and energy source limitations.

When computer graphics are used for entering and viewing data, application of the optimization program becomes straightforward. This addition makes the program more suitable for use by mill personnel and will promote the use of the optimization program in pulp and paper mills.

As a final thought, consider the waste of not taking this opportunity to reap the benefits of improved mill operation; the most important proposed future work is the
application of a schedule optimization method in real mills on a day to day basis.
REFERENCES


BAZ77 Bazaraa, S., Jarvis, J., Linear Programming and Network Flows, John Wiley & Sons, 1977


PAT80 Patterson, M., "Linear Programming", The University of British Columbia Computing Centre, pp. 16-17, Feb. 1980.


APPENDIX A

Optimization Program Listing and Flowcharts

Appendix A contains an explanation of the flowchart symbols used (Figure A1), the main program flowchart (Figure A2), the dynamic programming subroutine flowchart (Figure A3), and the program listing. The 'flow' of the cost function routine can be found in Figure 3.1.
Figure A1: Flowchart Symbols

i) Operation of statement(s) beginning at MTS file line number nn as shown in source listing

ii) Conditional statement

Yes
Statement is TRUE

No
Statement is FALSE

iii) DO LOOP number nn: index I from a to c in increments of b.
Figure A2: Successive Approximations, Multi-pass Dynamic Programming, Main Program

Start

Read:
- mill model and related information,
- number of iterations desired, shutoff and rate change penalties, billing rates, constraints, and output production schedules.

Sum the energy requirements for output processes.

Initialize production schedules [INIT] (customized to the mill)

Select next process for consideration

Do single process dynamic programming [DP]

Have all processes been considered on this iteration?

Yes

No

Print iteration number and cost of current solution

Reduce range of allowed buffer levels

finished all iterations?

No

Yes

Print trajectories of buffer levels, process rates, and energy demands [POUT]

Stop

* INITR is the number of iterations per grid spacing
Figure A3: Dynamic Programming Subroutine (DP)

Correct buffer constraints:
1) correct "master" buffer constraints so that the constraint limit occurs if shutoffs are obtained,
2) correct "slave" buffer constraints so that no shutoff is blocked by insufficient or excessive buffer levels.

Find 2 grid points on each side of the master buffer trajectory [LIMITS]

Remove any grid points that violate slave buffer constraints [SCHECK]

Sum steam and electrical demands of all but the process being considered

Find grid points for time k [LIMITS]

Remove grid points that violate slave buffer constraints [SCHECK]

Find control required to make transition from grid J(time k) to grid point L(time k+1).

Find cost of this trajectory (from time k to final time)

Continued...
Figure A3 continued

A

401

Is this the cheapest route yet for grid j?

---

Yes

Store this cost and route.

No

---

B

C

420

Save time k values for use with rate change penalties

D

E

433

Update process rate and associated buffer level trajectories

Return

458
Delta U method of Dynamic Programming with multi-pass, successive approximations, Version 3

82/2/4 S. Craig

DIMENSION ITORD(19)
COMMON /CFD/ COEE(4,30), COES(4), TSM1(30), TSM2(30), TEM(30), EFF, TGM(30)
COMMON /COSTPT/ ELEC, TG, CHNGF(30), PEN(19,30), NC1, NCM, CHNSTM
COMMON /DPOUT/ NST1, UFX(2,30), COST, U(20,30), M
COMMON /DPD/ CSU(2,19)
COMMON /CSDPT/ NSTAGE, STEAM, IU, SUBSTM, SUBELC
COMMON /SCHLIM/ CSX(31,2,15), RITRN, NSTATE
COMMON /TRJ/ XTRAJ(31,15), NCL1, BDF(15,20)
COMMON /VARD/ UFAC(2,20), ISLAVE(5,19), BDN(5,19), NSLAVE(19)

************ variable definitions ********
BDF --> full model interconnection matrix,
XTRAJ(k+1) = XTRAJ(k) + BDF * U(k)
BDN --> BDN(j,i) * trial delta X for process i = how the jth slave tank will vary for this trial delta X.

CFAC --> constraint closure rate
CHNGF --> process rate-change penalty
CHNSTM --> steam demand-change penalty
COEE --> electrical billing rate parameters
COES --> steam cost parameters
COST --> cost of iteration
CSU --> process (control,U) constraints
CSX --> tank level constraints
EFF --> turbo-generator efficiency
ELEC --> purchased electrical demand
INITR --> "inner iterations", number of iteration s with the same value of RITRN
IOPT --> = 0 for bypassing debug print-outs
= 0 for including debug print-outs
42 C ISLAVE --> points to the tanks associated with a process
43 C IST --> loop counter
44 C ITPNT --> points to state to vary
45 C ITR --> # of iterations to do in total
46 C ITSRT --> points to starting process for iterations
47 C IU --> is the number of the control (process) to vary
48 C M --> number of controls (processes), including O/P processes
49 C NCL --> number of non-O/P processes
50 C NCL1 --> NCL + 1
51 C NCM --> NCL - 1
52 C NSLAVE(I) --> # of tanks associated with process I
53 C NSTAGE --> # of time stages to consider
54 C NSTATE --> # of states (tanks)
55 C NST1 --> NSTAGE + 1
56 C PEN --> process rate penalties, used to shut off a process
57 C RITRN --> fraction of total constraint range being considered
58 C STEAM --> steam demand
59 C SUBELC --> electrical demand with the exception of one process's requirements
60 C SUBSTM --> as SUBELC but for steam demands
61 C TEM --> points to the demand level at which peak electrical demand begins (in steam equivalent units... i.e. demand divided by turbo-generator efficiency).
62 C TG --> turbo-generator steam flow
63 C TGM --> maximum electrical O/P of turbo-generator
64 C TSM1 --> steam demand at which the incremental cost of producing steam equals the base rate for purchased electrical power.
65 C TSM2 --> as TSM1 but for peak purchased electrical rate
66 C U --> control (process) trajectories
67 C UFAC --> steam and electrical usage factors for each process
68 C UFX --> summed steam and electrical demands for all specified O/P processes at each time stage
69 C XTRAJ --> tank level trajectories
ITR
READ (5,10) NSTAGE, NSTATE, NUMF, ITR, M, IOPT, IN
NST1 = NSTAGE + 1
READ (5,10) (ITORD(I), I=1,NCL)
READ (5,10) (NSLAVE(I), I=1,NCL)
DO 50 I = 1, NCL
NS = NSLAVE(I)
READ (5,10) (ISLAVE(J,I), J = 1,NS)
CONTINUE
DO 60 I = 1, NCL
NS = NSLAVE(I)
READ (5,40) (BDN(J,I), J=1,NS)
CONTINUE
READ (5,20) CFAC, RITRN, EFF, CHNSTM
READ (5,20) (UFAC(1,I), I=1,M)
READ (5,20) (UFAC(2,I), I=1,M)
READ (5,20) (CSU(1,I), I=1,NCL)
READ (5,20) (CSU(2,I), I=1,NCL)
READ (5,20) (XTRAJ(1,I), I=1,NSTATE)
DO 70 I = 1, NSTATE
READ (5,20) (BDF(I,J), J=1,M)
CONTINUE
DO 80 I = 1, NSTAGE
READ (5,20) TSM1(I), TSM2(I), TEM(I), TGM(I)
READ (5,20) (PEN(J,I), J=1,NCL), CHNGF(I)
READ (5,30) (COEE(U,I), J=1,4)
CONTINUE
READ (5,30) (COES(I), I=1,4)
DO 90 I = 1, NST1
READ (5,20) (CSX(I,1,J), J=1,NSTATE)
READ (5,20) (CSX(I,2,J), J=1,NSTATE)
CONTINUE
calculate the steam and electrical requirements of all specified O/P processes

DO 110 I = 1, NSTAGE
READ (5,20) (U(J,I),J=NCL1,M)
UFX(1,I) = 0.
UFX(2,I) = 0.
DO 100 J = NCL1, M
UFX(1,I) = UFAC(1,J) * U(J,I) + UFX(1,I)
UFX(2,I) = UFAC(2,J) * U(J,I) + UFX(2,I)
100 CONTINUE
110 CONTINUE

initialize the tank and process trajectories
CALL INIT
C debug print out
IF (IOPT .EQ. 0) GO TO 130
DO 120 I = 1, NSTAGE
WRITE (6,20) (XTRAJ(I,J),J=1,NSTATE)
WRITE (6,20) (U(J,I),J=1,M)
120 CONTINUE
WRITE (6,20) (XTRAJ(NST1,J),J=1,NSTATE)
ITR = 1
ITSRT = 1
main loop starts here:
DO 160 I = 1, INITR
IST = 1
ITPNT = ITSRT
IU = ITORD(ITPNT)
171  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
172  C    do dynamic programming with only one process free
173  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
174  C    to vary
175  C
176          CALL DP
177          IST = IST + 1
178          ITPNT = ITPNT + 1
179          IF (ITPNT .GT. NCL) ITPNT = 1
180          IF (IST .LE. NCL) GO TO 150
181          ITSRT = ITSRT + 1
182          IF (ITSRT .GT. NCL) ITSRT = 1
183          160 CONTINUE
184  C
185          WRITE (6,170) ITRN, COST
186          170 FORMAT (' ITERATION ', I4, ' COST ', 1PE12.5)
187  C
188          CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
189          C    reduce grid size by reducing RITRN
190          CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
191  C
192          ITRN = ITRN + 1
193          IF (RITRN .GT. .02) RITRN = RITRN * CFAC
194          IF ((IOPT .EQ. 0) .AND. (ITRN .LE. ITR)) GO TO 140
195          C
196          C    print trajectories
197  C
198          CALL POUT
199  C
200          IF (ITRN .LE. ITR) GO TO 140
201          STOP
202          END
203  C
204          SUBROUTINE DP
205  C
206          CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
207  C    This subroutine performs one pass of dynamic pro-
208  C    gramming w.r.t. one control (process,... U variable)
209  C
210          82/2/4 S. Craig
211  C
212          CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
213  C    ***** new variables *****
214  C    CSXT --> temporary, corrected subset of CSX
C IXP(K,I) --> points to next best state value for the Ith state value at time K
C RJK(I) --> cost of Ith possible state value at time K
C RJK1(I) --> as RJK(I) but at time K+1
C STMK(I) --> steam required for Ith possible state value at time K
C STMK1(I) --> as STMK(I) but for time K+1
C U0(I) --> control associated with possible state I, time K
C U02(I) --> as U0(I), but for time K+1
C XP(K,I) --> value of Ith possible state at time K

DIMENSION RJK(5), STMK(5), U0(5), IXP(50,5), RJK1(5)
COMMON /DPOUT/ NST1, UFX(2,30), COST, U(20,30), M
COMMON /DPD/ CSU(2,19)
COMMON /CSDPT/ NSTAGE, STEAM, IU, SUBSTM, SUBELC
COMMON /TRJ/ XTRAJ(31,15), NCL1, BDF(15,20)
COMMON /VARD/ UFAC(2,20), ISLAVE(5,19), BDN(5,19), NSLAVE(19)
COMMON /COSTPT/ ELEC, TG, CHNGF(30), PEN(19,30), NCL, NCM, CHNSTM
COMMON /POSX/ XP(31,5), U02(5), STMK1(5), CSXT(31,2,5)
COMMON /SCHLIM/ CSX(31,2,15), RITRN, NSTATE

initialize all costs to 0.0

DO 10 I = 1, 5
RJK1(I) = 0.
10 CONTINUE

NS = # of tanks "slave" to process IU
IUM = the tank # for the "master" (controlling) tank for this process (process IU)
DO MASTER TANK CASE FIRST
C  CSXT is loaded from CSX and modified so that shut
offs are
possible, where desired
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCC
C
I = 1
IF (BDF(ISM,IU) .LT. 0.) I = 2
XL = 0.
CSXT(1,1,1) = CSX(1,1,ISM)
CSXT(1,2,1) = CSX(1,2,ISM)
DO 40 K1 = 2, NST1
K = K1 - 1
CSXT(K1,1,1) = CSX(K1,1,ISM)
CSXT(K1,2,1) = CSX(K1,2,ISM)
IF (PEN(IU,K) .GT. 0.) GO TO 20
XL = 0.
GO TO 40
20 XL = XL - BDF(ISM,IU) * U(IU,K)
CSXT(K1,1,1) = XTRAU(K1,ISM) + XL
IF (I .EQ. 1) GO TO 30
CSXT(K1,2,1) = AMIN1(CSXT(K1,2,1),CSX(K1,2,ISM))
GO TO 40
30 CSXT(K1,1,1) = AMAX1(CSXT(K1,1,1),CSX(K1,1,ISM))
40 CONTINUE
C now do the remaining "slave" tank constraints
DO 190 IX = 1, NS
ITL = 0
ITH = 0
XL = 0.
XH = 0.
L = ISLAVE(IX,IU)
IF (IX .EQ. 1) GO TO 50
CSXT(NST1,1,IX) = CSX(NST1,1,L)
CSXT(NST1,2,IX) = CSX(NST1,2,L)
50 DO 150 KK = 1, NSTAGE
K = NST1 - KK
K1 = K + 1
NH = 0
NL = 0
IF (IX .EQ. 1) GO TO 60
CSXT(K,1,IX) = CSX(K,1,L)
CSXT(K,2,IX) = CSX(K,2,L)
60 DO 110 I = 1, NCL
IF ((I .EQ. IU) .OR. (PEN(I,K) .EQ. 0.)) GO
TO 110
IF (BDF(L,I)) 70, 110, 90
C TANK IS BEFORE SHUTDFF
70 IF (ITH .NE. 0) GO TO 80
ITH = K1
XH = CSXT(ITH,2,IX)
80   XH = XH + BDF(L,I) * U(I,K)
304  NH = 1
305  GO TO 110
306  C    TANK IS AFTER SHUTOFF
307  90   IF (ITL .NE. 0) GO TO 100
308  ITL = K1
309  XL = CSXT(ITL,1,IX)
310  100  XL = XL + BDF(L,I) * U(I,K)
311  NL = 1
312  110  CONTINUE
313  C
314  C assign modified values to CSXT if needed
315  C
316  IF ((NL .NE. 0) .OR. (ITL .EQ. 0)) GO TO 130
317  DD 120 I = K1, ITL
318  CSXT(I,1,IX) = AMIN1(XL,((CSX(I,2,L) - CSX(I,1,L))/2.))
319  120  CONTINUE
320  ITL = 0
321  XL = 0.0
322  C
323  130  IF ((NH .NE. 0) .OR. (ITH .EQ. 0)) GO TO 150
324  DD 140 I = K1, ITH
325  CSXT(I,2,IX) = AMAX1(XH,((CSX(I,2,L) - CSX(I,1,L))/2.))
326  140  CONTINUE
327  ITH = 0
328  XH = 0.0
329  C
330  150  CONTINUE
331  C
332  IF (ITL .EQ. 0) GO TO 170
333  DD 160 I = 2, ITL
334  CSXT(I,1,IX) = AMIN1(XL,((CSX(I,2,L) - CSX(I,1,L))/2.))
335  160  CONTINUE
336  C
337  170  IF (ITH .EQ. 0) GO TO 190
338  DD 180 I = 2, ITH
339  CSXT(I,2,IX) = AMAX1(XH,((CSX(I,2,L) - CSX(I,1,L))/2.))
340  180  CONTINUE
341  190  CONTINUE
342  C
343  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
344  C find 5 values of master tank levels as possible levels to
345  C consider at-time NST1. Check the corresponding va
346  C slave tanks to ensure constraints are not violate d
CALL LIMITS(ISM, NST1)
CALL SCHECK(NP2, NST1, NS)

>>>>> main loop of DP starts here <<<<<<

DO 240 KK = 1, NSTAGE
   K = NST1 - KK
   K1 = K + 1

SUBSTM = (-UFAC(1,IU)) * U(IU,K) + UFX(1,K)
SUBELC = (-UFAC(2,IU)) * U(IU,K) + UFX(2,K)

DO 200 I = 1, NCL
   SUBSTM = SUBSTM + UFAC(1,I) * U(I,K)
   SUBELC = SUBELC + UFAC(2,I) * U(I,K)

DCNTR = (XTRAJ(K,ISM) - XTRAJ(K1,ISM)) * BDN(1,I)

NFIND = 1

CALL LIMITS(ISM, K)
CALL SCHECK(NP1, K, NS)
DCNTR = (XTRAJ(K,ISM) - XTRAJ(K1,ISM)) * BDN(1,I)

U)
DO 220 J = 1, NP1
RJK(J) = 1E30
IFDFLG = 0
DO 210 L = 1, NP2
CF = 0.
C find the control to produce the change in tank level
C under consideration
UTRIAL = U(IU,K) + BDN(1,IU) * (XP(K1,L) - XP(K,d)) + DCNTR
IF (UTRIAL .LT. CSU(IU)) .OR. (UTRIAL .GT. CSU(2,IU))
1 CF = 1E3 * AMAX1((CSU(1,IU) - UTRIAL),(UTRIAL - CSU(2,IU)))
2 CF = CF + 1E2
C find the cost of producing/purchasing the required amount of energy
CF = CF + COSTF(UTRIAL,K,L) + RJK1(L)
IF (CF .GE. RdK(d)) GO TO 210
IFDFLG = 1
RJK(d) = CF
STMK(d) = STEAM
IXP(K,NFIND) = L
UO(d) = UTRIAL
210 CONTINUE
IF (IFDFLG .EQ. 0) GO TO 220
RJK(NFIND) = RJK(J)
XP(K,NFIND) = XP(K,J)
NFIND = NFIND + 1
220 CONTINUE
NP2 = NFIND - 1
DO 230 J = 1, NP2
RJK1(J) = RJK(J)
STMK1(J) = STMK(J)
UO2(J) = UO(J)
230 CONTINUE
CONTINUE
C save values at time k, for use at time K-1 with rate change penalties
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
NP2 = NFIND - 1
DO 230 J = 1, NP2
RJK1(J) = RJK(J)
STMK1(J) = STMK(J)
UO2(J) = UO(J)
230 CONTINUE
CONTINUE
C
C update process-rate and slave tank level trajectories
C
L = 1
DO 260 K1 = 2, NST1
K = K1 - 1
L1 = IXP(K,L)
U(IU,K) = U(IU,K) + BDN(1,IU) * (XTRAJ(K,ISM) -
1 XP(K1,L1) - XP(K,L))
L = L1
IF (NS .EQ. 1) GO TO 260
DCNTR = XP(K1,L1) - XTRAJ(K1,ISM)
DO 250 I = 2, NS
J = ISLAKE(I,IU)
XTRAJ(K1,  U) = XTRAJ(K 1 ,J) + BDN(I,IU) * DCNTR
CONTINUE
250 260 CONTINUE
C
CC update master tank trajectory
CC
L = 1
DO 270 K1 = 2, NST1
L = IXP(K1 - 1,L)
XTRAJ(K1,ISM) = XP(K1,L)
270 CONTINUE
C
COMMON /SCHLIM/ CSX(31,2,15), RITRN, NSTATE
C
SUBROUTINE LIMITS(I, K)
C
This routine finds 5 points, 2 on each side of the current
trajectory and within the constraints given by CS
XT. The
distance from the current trajectory (XTRAJ) is governed
by the constraint closure value "RITRN".
COMMON /SCHLIM/ CSX(31,2,15), RITRN, NSTATE
471 COMMON /TRJ/ XTRAJ(31,15), NCL1, BDF(15,20)
472 COMMON /POSX/ XP(31,5), U02(5), STMK1(5), CSXT(31, 2,5)
473 C
474 DEL = (CSXT(K,2,1) - CSXT(K,1,1)) * RITRN
475 XP(K,3) = XTRAJ(K,IST)
476 XP(K,5) = XP(K,3) + DEL
477 IF (XP(K,5) .GT. CSXT(K,2,1)) XP(K,5) = CSXT(K,2,1  )
478 XP(K,1) = XP(K,3) - DEL
479 IF (XP(K,1) .LT. CSXT(K,1,1)) XP(K,1) = CSXT(K,1,1)
480 XP(K,4) = (XP(K,5) + XP(K,3)) / 2.
481 XP(K,2) = (XP(K,1) + XP(K,3)) / 2.
482 RETURN
483 END
484 C
485 SUBROUTINE SCHECK(N, K, NS)
486 C
487 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
488 C The points produced by "LIMITS" are checked to ensure that
489 C none of the XP force a slave tank out of bounds.
490 C If a XP
491 C does force another tank out of bounds, that XP is
492 C from the list of possibilities.
493 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
494 C
495 COMMON /CSDPT/ NSTAGE, STEAM, IU, SUBSTM, SUBELC
496 COMMON /TRJ/ XTRAJ(31,15), NCL1, BDF(15,20)
497 COMMON /VARD/ UFAC(2,20), I SLAVE(5,19), BDN(5,19),
498 COMMON /POSX/ XP(31,5), U02(5), STMK1(5), CSXT(31, 2,5)
499 C
500 IF (NS .EQ. 1) RETURN
501 N = 0
502 CNTR = XP(K,3)
503 DO 20 I = 1, 5
504 DEL = XP(K,I) - CNTR
505 DO 10 J = 2, NS
506 L = ISLAVE(J,IU)
507 XPOS = XTRAJ(K,L) + BDN(J,IU) * DEL
508 IF ((XPOS .GT. CSXT(K,2,J)) .OR. (XPOS .LT. CSXT(K,1,J))
509 1 GO TO 20
510 CONTINUE
511 N = N + 1
XP(K,N) = XP(K,I)
20 CONTINUE
RETURN
END

FUNCTION COSTF(U, K, L)
C
" find the cost for this control at time k
C
COMMON /CFD/ COEE(4,30), COES(4), TSM1(30), TSM2(3 O), TEM(30).
1  EFF, TGM(30)
COMMON /COSTPT/ EPWR, TG, CHNGF(30), PEN(19,30), N CL, NCM, CHNSTM
COMMON /COSTPT/ NSTAGE, STM, IU, SUBSTM, SUBELC
COMMON /VARD/ UFAC(2.20), ISLAVE(5,19), BDN(5,19), NSLAVE(19)
COMMON /POSX/ XP(31,5), UO2(5), STMK1(5), CSXT(31, 2,5)
C
COSTF = 0.
C
" assign penalty for process-rate change
C
IF (K .EQ. NSTAGE) GO TO 10
COSTF = CHNGF(K) * SQRT(ABS(UO2(L) - U))
C
" assign penalty for process operation
C
10 COSTF = COSTF + PEN(IU,K) * SQRT(ABS(U))
C
TG = 0.
SO = SUBSTM + U * UFAC(1,IU)
EPWR = SUBELC + U * UFAC(2,IU)
STM = SO
C
" FIND THE STEAM EQUIVALENT OF THE ELECTRICAL DEM
AND
555 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

556 C
557 EEQ = EPWR / EFF
558 C
559 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

560 C COMPARE INCRAMENTAL COSTS
561 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

562 C
563 IF (SO .GE. TSM2(K)) GO TO 30
564 SE = COEE(2,K).
565 IF (EEQ .GT. TEM(K)) SE = COEE(4,K)
566 SS = 0.
567 IF (SO .GE. TSM1(K)) SS = COEE(2,K)
568 IF (SS .GE. SE) GO TO 30
569 C
570 DIF = EEQ - TEM(K)
571 IF (DIF .LE. 0.) GO TO 20
572 C
573 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

574 C ELECTRICAL IS INTO PEAK DEMAND CHARGES
575 C REDUCE THIS PURCHASE REQUIREMENT AS MUCH AS POSSIBLE
576 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

577 C
578 TG = AMIN1((TSM2(K) - SO),TGM(K),DIF)
579 STM = SO + TG
580 C
581 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

582 C ADJUST ELECTRICAL TO REFLECT EXTRA STEAM PRODUCTI
583 CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

584 C
585 EEQ = EEQ - TG
586 IF ((STM .EQ. TSM2(K)) .OR. (TG .EQ. TGM(K))) GO T
587 O 30
588 C
589 DIF = TSM1(K) - STM
590 IF (DIF .LE. 0.) GO TO 30
591 C
592 C BRING STEAM UP TO LOWER INCRAMENTAL COST OF ELECT
593 C IF NEEDED
594  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CCCCCCCCCC
595  C
596  SS = TGM(K) - TG
597  SO = AMIN1(EQQ,DIF,SS)
598  STM = STM + SO
599  EEQ = EEQ - SO
600  TG = TG + SO
601  C
602  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CCCCCCCCCC
603  C    FIND ADJUSTED ELECTRICAL DEMAND (FROM B.C. HYDRO)
604  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CCCCCCCCCC
605  C
606  30   EPWR = EEQ * EFF
607  C
608  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CCCCCCCCCC
609  C    FIND COST OF STEAM
610  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CCCCCCCCCC
611  C
612  IF (K .EQ. NSTAGE) GO TO 40
613  SO = STM - STMK1(L)
614  COSTF = COSTF + CHNSTM * SO * SO
615  40   COSTF = COES(1) + STM * (COES(2) + STM*(COES(3) +
      COES(4)*STM)) +
616     COSTF
617  C
618  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CCCCCCCCCC
619  C    FIND COST OF ELECTRICAL POWER
620  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CCCCCCCCCC
621  C
622  IF (EEQ .GT. TEM(K)) GO TO 50
623  COSTF = COSTF + COEE(1,K) + COEE(2,K) * EPWR
624     RETURN
625  C
626  50   COSTF = COSTF + COEE(3,K) + COEE(4,K) * EPWR
627     RETURN
628     END
629  C
630  SUBROUTINE POUT
631  C
632  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CCCCCCCCCC
633  C    This routine prints the: process rate, tank level
634  C    steam demand, turbo-generator production, and purchased
635  C    electrical power trajectories.
The cost is calculated with no penalties for process rate changes, process rates or steam demand fluctuations.

COMMON /COSTPT/ ELEC, TG, CHNGF(30), PEN(19,30), NCL, NCM, CHNSTM
COMMON /DPOUT/ NST1, UFX(2,30), COST, U(20,30), M
COMMON /CSDPT/ NSTAGE, STEAM, IU, SUBSTM, SUBELC
COMMON /TRd/ XTRAd(31,15), NCL, BDF(15,20)
COMMON /VARD/ UFAC(2,20), ISLAVE(5,19), BDN(5,19), NSLAVE(19)
COMMON /POSX/ XP(31,5), U02(5), STMK1(5), CSXT(31,2,5)
COMMON /SCHLIM/ CSX(31,2,15), RITRN, NSTATE

RS = CHNSTM
CHNSTM = 0.
COST = 0.
IU = NCL
DO 40 K = 1, NSTAGE
  SUBSTM = steam demand of all but last unspecified process
  SUBELC = as SUBSTM but for electrical demand

SUBSTM = UFX(1,K)
SUBELC = UFX(2,K)
DO 10 I = 1, NCM
  SUBSTM = SUBSTM + UFAC(1,I) * U(I,K)
  SUBELC = SUBELC + UFAC(2,I) * U(I,K)
10 CONTINUE
RP = PEN(NCL,K)
RC = CHNGF(K)
CHNGF(K) = 0.
PEN(NCL,K) = 0.
UINTRIAL = U(NCL,K)
COST = COST + COSTF(UINTRIAL,K,1)
PEN(NCL,K) = RP
CHNGF(K) = RC
WRITE (6,20) K, (XTRAd(K,I),I=1,NSTATE)
20 FORMAT (13,15F7.3)
WRITE (6,30) (U(I,K),I=1,M), STEAM, TG, ELEC
30 FORMAT (23F6.3)
40 CONTINUE
WRITE (6,20) NST1, (XTRAd(NST1,I),I=1,NSTATE)
WRITE (6,50) COST
50 FORMAT (15X,'COST=',1PE12.5)
CHNSTM = RS
681 RETURN  |  682 END
This Appendix presents a brief analysis of the relative computation times required for different numbers of grid points, with successive approximations, multi-pass dynamic programming. The approach used in the analysis is to find the number of iterations needed to meet 'state space coverage' and final grid spacing requirements. Whether or not the resulting iteration parameters permit convergence is not dealt with in this approach. The majority of the calculations occur during computation of the cost function. The number of cost function calculations is proportional to the square of the number of grid points, times the number of iterations; therefore, a work index (WI) is defined as:

\[ WI = n^2 i, \]

where \( n \) is the number of grid points, and \( i \) is the number of iterations.

The final grid spacing, \( d \), is given by:

\[ d = \frac{2ar^{i-1}}{(n-1)} \]  \hspace{1cm} (B-1)

where: \( r \) is the rate at which the constraint range is reduced, and \( a \) is one half the fractional range of the constraints covered by the starting grid.

Using the identity:

\[ ar^0 + ar^1 + ar^2 + \ldots + ar^{i-1} = a(1-r^i)/(1-r), \quad r \neq 1, \]

the maximum fraction of the total state space range that can
be covered, \( c \), is:

\[ c = \frac{a(1-r^i)}{(1-r)}. \quad (B-2) \]

Solving for \( r \) in \( B-1 \) and substituting this expression for \( r \) into \( B-2 \) yields:

\[ r = \frac{a-c}{(d(n-1)/2)-c}. \quad (B-3) \]

From \( B-1 \):

\[ i = 1 + \frac{\ln(d(n-1)/2a)}{\ln(r)}. \]

Using \( B-3 \) for \( r \), gives:

\[ i = 1 + \frac{\ln(d(n-1)/2a)}{\ln((a-c)/(.5*d(n-1)-c))}. \quad (B-4) \]

Equations \( B-4 \) and \( B-3 \) were used to produce the plots of Figure B1 which show the work index (WI) and constraint closure rate (\( r \)) as a function of the number of grid points (\( n \)) and initial state space coverage (\( a \)). The total state space coverage parameter, \( c \), was set to 1 (for complete coverage) and a final grid spacing of 0.02 was used for all the cases shown in Figure B1. The plots of Figure B1 indicate that, for the criteria used in this analysis, less computation is required when a large initial grid spacing and few grid points are used.
Figure B1: Computational Work Index versus Initial Grid Spacing and Number of Grid Points and Constraint Closure Rate versus Initial Grid Spacing and Number of Grid Points
APPENDIX C

Graphical Input Program Listing and Flowcharts

Appendix C contains flowcharts of the main program for the graphical input of data (Figure C1), flowcharts for some of the subroutines called by the main program (Figures C2 and C3), and the program listing. See Figure A1 for an explanation of the flowchart symbols used.
Figure C1: Graphical Input Program (Main Program)

Start

1. Read the first line of the input data file in order to determine the file's dimensions

2. Read the input file

3. Define grid to be used with all graphical input

4. Display menu of file sections to be altered

5. Read operator's selection from keyboard

6. A. Production rates

7. B. Buffer constraints

8. C. Rate penalties

9. D. Rate-change penalties

10. E. Turbogenerator limits

11. Continue

12. Read initial conditions

13. Make any requested alterations to electrical billing parameters

14. Write modified input data back to the input file

15. Stop

Continued....
Figure C1 continued

A

116
Read process number from keyboard

Graphically update process rate trajectory [GIP]

B

134
Read buffer number from keyboard

Display green grid

Label vertical axis [LBLY]

Display lower constraint in red [DLN]

Display upper constraint in red [DLN]

Update upper constraint values [CURIN]

Display blue verification curve [DLN]

Update lower constraint values [CURIN]

Display blue verification curve [DLN]

C

160
Read process number from keyboard

Graphically update penalty trajectory [GIP]

D

175
Graphically update rate-change penalty [GIP]

E

184
Graphically update turbogenerator limit [GIP]
Figure C2: Graphical Input Program (Subroutine GIP)

Start
Display green grid
Label vertical axis [LBYL]
Draw current trajectory in red [DLN]
Read new trajectory values from cursor position [CURIN]
Draw blue curve to verify new trajectory [DLN]
Return

Figure C3: Graphical Input Program (Subroutine CURIN)

Start
Read key and cursor position
key 'S'
key 'V'
key 'B'
key 'E'

Display error message [ERMS]
Update data vector
Update data vector
Linearly interpolate the points in the section to be changed
Save cursor position as the beginning of a data entry
Display the cursor coordinates
Return

Is there an entry needing ending?
Continuous type of data?
Graphical input program, for use with data files for the DP program.

This program is to run by:

RUN GIP.C+LIG:COLLIB 3=error file 7=data.I/P.f

>>>>>>>> run on the TEKTRONICS 4027 only <<<<<<

The basic operation of this program is to read in the data file, display the requested segments of the file, and make changes to these segments using I/P from the CURSOR, then writing the I/P data back to the file.

Variable definitions:

ARMAX --> maximum tank levels and maximum T/G rate
ARMIN --> minimum
CHNGF --> process rate change penalty
COEE --> cost coefficients for electrical purchase
CSU --> lower + upper limits on process rates
CSX1 --> lower constraints on tank levels (not absolute constraints)
CSX2 --> upper
IANS --> answer set for keyboard responses.
PEN --> process shut-off penalty
TEM --> electrical demand pt. at which peak charge starts
TGM --> maximum O/P of T/G
TSM1 --> pt. where steam incremental cost = base electrical charge
TSM2 --> peak
UROP --> production schedule
XT1 --> initial tank levels

note NOTE note NOTE note NOTE note NOTE note NOTE note NOTE note NOTE note NOTE note
C FORTRAN WRITES to L.U.N. 6 of "!..." are TEKTRONIC S 4027 instructions
C See the TEKTRONICS Programmer's reference guide for more information.
C

* GRAPHICAL GRIDDIA;
* GRIDDIA := BLANK;
IDENTER IANS(6)
DIMENSION XT1(15),CSX1(15,30),CSX2(15,30),
1 UPROD(7,30),PEN(20,30),
2 CHNGF(30),C0EE(4,30),TSM1(30),TSM2(30),C
OES(4),
3 TEM(30),TGM(30),CSU(20,2),ARMIN(16),ARMA
X(16)
DATA IANS/'S','V','B','E','Y','N'/
GRN=240.
YLW=180.
RED=120.
BLUE=300.

C
C************ read the I/P file
C************
C
READ(7,100) NSTAGE,NSTATE,NUMF,I,M
NCL=M-NUMF
NST1=NSTAGE+1
LNCT=(4+2*NCL)*1000
LN1=LNCT
READ(7'LNCT,110) R,R1,EFF
110 FORMAT(23F6.3)
100 FORMAT(1513)
LNCT=LNCT+3000
LN2=LNCT
READ(7'LNCT,110) (CSU(I,1),I=1,M)
READ(7,110) (CSU(I,2),I=1,M)
LNCT = LNCT+ 1000*(3+NSTATE )
FIND(7'LNCT)
LN3=LNCT
DO 10 I=1,NSTAGE
   READ(7,110) TSM1(I),TSM2(I),TEM(I),TGM(I)
   READ(7,110) (PEN(J,I),J=1,NCL),CHNGF(I)
   READ(7,120) (C0EE(J,I),J=1,4)
10 CONTINUE
120 CONTINUE
READ(7,120) (C0ES(I),I=1,4)
DO 20 I=1,NST1

C
READ(7,110) (CSX1(J,I),J=1,NSTATE)
READ(7,110) (CSX2(J,I),J=1,NSTATE)
CONTINUE
C
LNCT=LNCT+1000*(NSTAGE*3+NST1*2+1)
FIND(7'LNCT)
LN4=LNCT
DO 30 I=1,NSTAGE
READ(7,110) (UPROD(J,I),J=1,NSTAGE)
30 CONTINUE
01=NSTATE+1
READ(7,110) (ARMIN(I),I=1,J1)
READ(7,110) (ARMAX(I),I=1,J1)
C*******************************************************
***********
C I/P SECTION
C get operator's request for area of interest:
C*******************************************************
***********
CALL GRID(NSTAGE)
DISPLAY '**************' AT 1.1.,1;
WRITE(6,130)
FORMAT(1X,/, '1-PROD. 2-TANKS 3-PEN. 4-CCHANGE
5-T/G 6-CONT.')
READ(5,200) I
IF((I.LT.1).OR.(I.GT.NUMF)) CALL ERMS(&40)
WRITE(6,141) I
FORMAT(1STR/ PRODUCTION OUTPUT #',13,/'')
J=M-NUMF+1
RMIN=CSU(J,1)
RMAX=CSU(J,2)
CALL GIP(UPROD,RMIN,RMAX,NSTAGE,RED,YLW,GRN,BLUE,I ANS,
1    7,30,I)
GO TO 40
C
C******************************************************************************

C TANK CONSTRAINTS:
C******************************************************************************

C
* 60: DISPLAY '**************' AT 1.1,.1;
WRITE(6,160)
160 FORMAT(' ENTER TANK # ')
READ(5,111) I
IF((I.LT.1).OR.(I.GT.NSTATE)) CALL ERMS(&40)
* DISPLAY GRIDDIA COLOUR GRN;
* DRAW FROM .3,.93 TO .3,.92;
WRITE(6,161) I
161 FORMAT('!STR /TANK #',13,'/')
RMIN=ARMIN(I)
RMAX=ARMAX(I)
CALL LBLY(RMIN,RMAX)
CALL DLN(RMIN,RMAX,CSX1,RED,NSTAGE,1,15,31,I)
CALL DLN(RMIN,RMAX,CSX2,RED,NSTAGE,1,15,31,I)
* DISPLAY 'UPPER CONSTRAINT' AT .7,.92;
CALL CURIN(NSTAGE,CSX2,RMIN,RMAX,IANS,YLW,1,15,31)
* CALL DLN(RMIN,RMAX,CSX2,BLUE,NSTAGE,1,15,31,I)
* DISPLAY 'LOWER' AT .7,.92;
CALL CURIN(NSTAGE,CSX1,RMIN,RMAX,IANS,YLW,1,15,31,
I)
CALL DLN(RMIN,RMAX,CSX1,BLUE,NSTAGE,1,15,31,I)
GO TO 40
C
C******************************************************************************

C PENALTIES
C******************************************************************************

C
70 WRITE(6,170)
170 FORMAT(' ENTER PROCESS # ')
READ(5,111) I
IF((I.LT.1).OR.(I.GT.NCL)) CALL ERMS(&40)
* DRAW FROM .25,.93 TO .25,.92;
WRITE(6,171) I
171 FORMAT('!STR /PENALTY FOR PROCESS #',13,'/')
CALL GIP(PEN,0.,99.,NSTAGE,RED,YLW,GRN,BLUE,IANS,
1 20,30,I)
GO TO 40
C
C******************************************************************************

C PROCESS CHANGE PENALTY:
C******************************************************************************
174 C
175 * 80: DISPLAY 'PROCESS CHANGE PENALTY' AT .3,.92;
176 CALL GIP(CHNGF,.0.,20.,NSTAGE,RED,YLW,GRN,BLUE,IALNS
177 , 1 1,30,1)
178 GO TO 40
179 C
180 C*********************************************************************** ***********
181 C  T/G MAXIMUM
182 C*********************************************************************** ***********
183 C
184 * 90: DISPLAY 'TURBO-GENERATOR LIMITS' AT .3,.92;
185 RMAX=ARMAX(NSTATE+1)
186 CALL GIP(TGM,.0.,RMAX,NSTAGE,RED,YLW,GRN,BLUE,IALNS
187 , 1 1.30,1)
188 GO TO 40
189 C
190 C*********************************************************************** ***********
191 C  GET INITIAL CONDITIONS:
192 C*********************************************************************** ***********
193 C
194 210 WRITE(6,180)
195 180 FORMAT('!WOR 0',/,' ENTER INITIAL TANK LEVELS ')
196 READ(5,111) (XT 1(I),I = 1,NSTATE)
197 111  FORMAT(20G10.3)
198 DO 83 I=1,NSTATE
199 CSX1(I, 1)=XT1(I )
200 CSX2(1,1)=XT1(1)
201 83 CONTINUE
202 WRITE(6,192)
203 192 FORMAT(' ELECTRICAL CHARGE PARAMETER ENTRY SECTION : ')
204 C
205 C*********************************************************************** ***********
206 C  FIND ELECTRICAL COEFFICIENTS FOR COST FUNCTION:
207 C*********************************************************************** ***********
208 C
209 I1=0
210 IF(COES(4).EQ.0) GO TO 91
211 I1=1
212 A3=COES(4)*3.
213 BDA=-COES(3)/A3
214 BSD= COES(3)*COES(3)
215 91 WRITE(6,190)
216 190 FORMAT(' ENTER TIME RANGE FOR COST PAR.'S , -VE T
DO STOP')
    READ(5,111) I,J
    IF(I.LT.1) GO TO 201
    IF((I.GT.J).OR.(J.GT.NSTAGE)) GO TO 91
    WRITE(6,191)
    191 FORMAT(' ENTER RATE CHANGE POINT, BASE RATE, ',
    1 'PEAK RATE')
    READ(5,111) R,COEE(2,I),COEE(4,I)
    IF(COEE(2,I).GT.COEE(4,I)) GO TO 92
    COEE(3,I)=R*(COEE(2,I)-COEE(4,I))
    TEM(I)=R/EFF

C solve for points of equal Increamental cost, using the quadratic
C equation if necessary (positive real root taken)
C
C**********************************************************************

C**********************************************************************
   IF(I1.EQ.0) GO TO 93
   TSM1(I)=BDA+SQRT(BSD-A3*(COES(2)-COEE(2,I)))/A3
   TSM2(I)=BDA+SQRT(BSD-A3*(COES(2)-COEE(4,I)))/A3
   GO TO 94
   93 TSM1(I)=(COEE(2,I)-COES(2))/(2*COES(3))
   TSM2(I)=(COEE(4,I)-COES(2))/(2*COES(3))
   94 DO 95 L=I,J
      TSM1(L)=TSM1(I)
      TSM2(L)=TSM2(I)
      COEE(2,L)=COEE(2,I)
      COEE(3,L)=COEE(3,I)
      COEE(4,L)=COEE(4,I)
      TEM(L)=TEM(I)
   95 CONTINUE
   GO TO 91

C**********************************************************************

C WRITE BACK TO I/P FILE
C**********************************************************************

C 201 LNCT=LN2+2000
    WRITE(7,'(LNCT,11O) (XT1(I),I=1,NSTATE)
    FIND(7,'LN3)
    DO 202 I=1,NSTAGE
       WRITE(7,110) TSM1(I),TSM2(I),TEM(I),TGM(I)
       WRITE(7,110) (PEN(J,I),J=1,NCL),CHNGF(I)
       WRITE(7,120) (COEE(J,I),J=1,4)
   202 CONTINUE
    READ(7,120) (COE(I),I=1,4)
DO 203 I=1,NST1
   WRITE(7,110) (CSX1(J,I),J=1,NSTATE)
   WRITE(7,110) (CSX2(J,I),J=1,NSTATE)
203   CONTINUE
FIND(7,'LN4')
DO 204 I=1,NSTAGE
   WRITE(7,110) (UPROD(J,I),1,NUMF)
204   CONTINUE
C******************************************************* ************
C Since LIG resets the function keys on the 4027, this call to TEK
C sets them back to the definitions in the file TEK
C******************************************************* ************
CALL MTSCMD('SC0 TEK',7)
STOP
END
C******************************************************* ************
CURSOR I/P CALLING ROUTINE
C******************************************************* ************
SUBROUTINE GIP(ARRAY,RMIN,RMAX,NSTAGE,RED,YLW,GRN,BLUE,IANS,ID1,ID2,IR)
DIMENSION ARRAY(ID1,ID2)
INTEGER IANS(6)
*  DISPLAY '**************' AT 1.1, 1:
*  DISPLAY GRIDDIA COLOUR GRN:
CALL LBLY(RMIN,RMAX)
CALL DLN(RMIN,RMAX,ARRAY,RED,NSTAGE,O,ID1,ID2,IR)
CALL CURIN(NSTAGE,ARRAY,RMIN,RMAX,IANS,YLW,O,ID1,ID2,IR)
CALL DLN(RMIN,RMAX,ARRAY,BLUE,NSTAGE,O,ID1,ID2,IR)
RETURN
END
C******************************************************* ************
**C**

---

**C** Draw a line through the points in ARRAY. If J=0, then the plot is
discrete time, it J=1 the plot is continuous time

---

**C**

**DIMENSION** ARRAY(ID1, ID2)

**C**

**DIF** = .65/(RMAX - RMIN)
**DX** = .75/FLOAT(NS)
**X** = .25

**DO 10 I=1, NS**

**X1** = **X** + **DX**

**R** = (ARRAY(IR, I) - RMIN) * **DIF** + .2

**R1** = (ARRAY(IR, I+J) - RMIN) * **DIF** + .2

**DISPLAY** LINE FROM **X**, **R** TO **X1**, **R1** COLOUR **CLR**;

**X** = **X1**

**10 CONTINUE**

**RETURN**

**END**

**SUBROUTINE GRID(NS)**

---

**C**

---

**C** This routine makes a graph grid and time scale, common to all sections
of I/P. The grid is assigned to the LIG graphical variable GRIDDIA.

---

**C**

---

**C** Y grid:

---

**Y** = .2

**DO 10 I=1,11**

**GRIDDIA**:= GRIDDIA + **LINE** FROM .23,**Y** TO 1.02,**Y**; **Y** = **Y** + .065

**10 CONTINUE**

---
C X grid:

**********

X=.25
DX=.75/FLOAT(NS)
NS1=NS+1
DO 20 I=1,NS1
* GRIDDIA: GRIDDIA + LINE FROM X,.18 TO X,.87:
X=X+DX
20 CONTINUE
C

**********

C X axis label:

**********

X=.24
IDIF=INT(.25/DX)
DX=D*X*FLOAT(IDIF)
DO 40 I=1,NS1,IDIF
* GRIDDIA: GRIDDIA + IVALUE(I) AT X,.1;
X=X+DX
40 CONTINUE
C

**********

C TITLE ETC.:

**********

GRIDI"

**********

GRIDDIA: GRIDDIA + TVALUE('TIME') AT .55,.01
* TVALUE(=CURSOR') AT 1.1,9
+ TVALUE('POSITION') AT 1.1,.05
* TVALUE('MAGNITUDE') AT 1.1,.65
* TVALUE('TIME: ') AT 1.1,.75
* TVALUE('V - VERIFY') AT 1.1,.35
* TVALUE('B - BEGIN ENTRY ') AT 1.1,.3
* TVALUE('E - END ENTRY ') AT 1.1,.25;
RETURN
END
C

**********

SUBROUTINE LBLY(RMIN,RMAX)
C

**********

C LABELS, Y AXIS:

**********

C

**********

DIF=(RMAX-RMIN)/5.
Y=.2
V=RMIN
DO 30 I=1,6
* DRAW FROM .25,Y TO O.,Y;
WRITE(6,100) V
100 FORMAT('ISTR /.'F7.3/,')
V=V+DIF
30 CONTINUE
RETURN
C

**********

C This routine obtains new data for ARRAY via CURSOR
I/P

**********

C INTEGER IANS(6),KRLY
D DIMENSION ARRAY(ID1,ID2)
C
NS1=NS+1
IBFLAG=0
YDIF=RMAX-RMIN
DIF=.75/FLOAT(NS)
YLIG=.5
XLIG=.6
C

centre cursor, or return it to last spot
RETURN
C

**********

* 1O: DISPLAY LINE FROM (XLIG-.015),YLIG TO XLIG,YLIG;
CURSOR ON;
DISPLAY '************' AT 1.1,.1;
READ CURSOR XLIG,YLIG;
READ KEY KRLY;
IF(KRLY.EQ.IANS(1)) RETURN
C

**********

C Get "user" values from graphical position
C

**********

C
436 \ YU=((YLIG-.2)/.65)*YDIF+RMIN
437 IF(YU.GT.RMAX) YU=RMAX
438 IF(YU.LT.RMIN) YU=RMIN
439 IXU=INT((XLIG-.25)/DIF)+1
440 IF(IXU.LT.1) IXU=1
441 IF(IXU.GT.NS1) IXU=NS1
442 C
443 IF(KRLY.EQ.IANS(2)) GO TO 30
444 IF(KRLY.EQ.IANS(3)) GO TO 20
445 IF(KRLY.EQ.IANS(4)) GO TO 40
446 CALL ERMS(&10)
447 C
448 ******************************************************* ***********
449 C  SAVE START POSITIONS
450 ******************************************************* ***********
451 C
452 20 IBFLAG=1
453 YBLIG=YLIG
454 IB=IXU
455 BY=YU
456 C
457 ******************************************************* ***********
458 C  DISPLAY USER VALUES
459 ******************************************************* ***********
460 C
461 * 30: DRAW FROM 1.1, .71 TO 1.1, .7;
462 WRITE(6,110) IXU
463 110 FORMAT(''ISTR /,13,''/)
464 *  DRAW FROM 1.1, .61 TO 1.1, .6:
465 WRITE(6,120) YU
466 120 FORMAT('ISTR '/,F7.3,''/)
467 GO TO 10
468 C
469 ******************************************************* ***********
470 C  END ENTRY, CHANGE ARRAY:
471 C ******************************************************* ***********
472 C
473 40 IF(IBFLAG.NE.1) CALL ERMS(&10)
474 I1=MINO(IB,IXU)
475 I2=MAXO(IB,IXU)
476 IBFLAG=0
477 XI=FLOAT(IB-1)*DIF+.25
478 IF(J.NE.0) GO TO 70
479 DO 60 I=I1,I2
480 ARRAY(IR,I)=BY
481 60 CONTINUE
C
X2=FLOAT(IXU)*DIF+.25
IF(IXU.GE.IB) GO TO 61
X2=X2-DIF
X1=X1+DIF
* 61: DRAW FROM (X1+.01),YBLIG TO X1,YBLIG;
* DISPLAY LINE FROM X1,YBLIG TO X2,YBLIG COLOUR CLR
    GO TO 10
C
C********************************************************************************************************
C Continuous time type of data... linear interpolation may be needed
C********************************************************************************************************

C
70 ARRAY(IR,IB)=BY
ARRAY(IR,IXU)=YU
X2=FLOAT(IXU-1)*DIF+.25
IF(X2.EQ.X1) GO TO 90
* 100: DRAW FROM (X1+.01),YBLIG TO X1,YBLIG;
* DISPLAY LINE FROM X1,YBLIG TO X2,YBLIG COLOUR CLR;
I=I2-I1
IF(I.LT.2) GO TO 10
J1=I2-I1-1
XDIN=(YU-BY)/FLOAT(IXU-IB)
IF(IXU.LT.IB) BY=YU
DO 80 I=1,J1
    J2=I+I1
    ARRAY(IR,J2)=FLOAT(I)*XDIN+BY
80 CONTINUE
GO TO 10
C
C********************************************************************************************************
C Begin and end pt. are in the same time slot. Leave a small mark to acknowledge this.
C********************************************************************************************************

C
90 X1=X1-.005
X2=X2+.005
YBLIG=YLIG
GO TO 100
END
C
C
SUBROUTINE ERMS(*)
* DISPLAY 'INVALID ENTRY' AT 1.1.1:
RETURN
APPENDIX D

Graphical Output Program Listing and Flowchart

Appendix D contains the flowchart (Figure D1) and listing for the program used to graphically display the results of the optimization program. See Figure A1 for an explanation of the flowchart symbols used.
Figure D1: Graphical Output Program

Start

Read the first line of the optimization program's input file in order to determine characteristics of this and the optimum trajectory file.

Read the first line of the schematic diagram file in order to determine its dimensions

Read the optimum trajectory file

Read coordinates for $I^{th}$ oval, from schematic diagram file

Display aqua oval

Plot constraints in blue and levels for $I^{th}$ buffer in yellow within oval

Read coordinates for $I^{th}$ square from schematic diagram file

Display $I^{th}$ red square

Plot $I^{th}$ process rate within square

Draw lines and arrows to connect the processes (squares) and buffers (ovals).

Add energy plots if desired

Label display

Display menu of possible displays

Read operator's selection

continued...
Figure D1 continued

190

Stop?

Yes → Stop

No

191

Whole mill display?

Yes → 91

No

192

Single process display?

Yes

Read process number from keyboard

Plot process rate trajectory

90

No

193

Single buffer display?

Yes

Read buffer number from keyboard

Plot buffer level trajectory

Plot buffer level constraints

90

No

194

Steam, electrical, and turbogenerator demands?

Yes

Plot total steam demand, turbogenerator load, and purchased power demand

90

No
Graphical O/P program, for displaying the results of the DP program

To run this program (GOP.C):
RUN GOP.C+LIG:COLLIB 3=error file 7=trajectory file 8=graphical data 4=I/P file for DP program

Variable definitions

Graphical variables:
- ARROW --> an arrow head for pointers
- AXIS --> X,Y axis without labels
- LEVL --> a line representing one of the trajectories
- OVAL --> ellipse used to represent a storage tank
- TXT --> text string

Fortran variables:
- CSU --> process maximums
- EX --> array for steam,T/G and purchased power trajectories
- IANS --> keyboard I/P answer set
- IPRD --> reordering vector for tank trajectories
- ITEXT --> text string
- TMAX --> absolute maximum tank level
- TRAJ --> tank level trajectories
- UT --> process rate trajectories

**GRAPHICAL AXIS,ARROW,OVAL,LEVL,TXT;**
- AXIS :- BLANK;
- ARROW :- BLANK;
- OVAL :- BLANK;
- LEVL :- BLANK;
- TXT :- BLANK;

DIMENSION TRAJ(15,31),UT(20,30),IPRD(15),EX(3,30)
DIMENSION TMAX(15),CSU(20),CS1(15,31),CS2(15,31)
INTEGER*2 ITEXT(6)
INTEGER IANS(5)
DATA IANS/'E','T','P','W','S'/

define graphical variables
assign colours

RED=120.
YLW=180.
GRN=240.
AQUA=300.
BLUE=330.

read # of states (tanks), # of controls (processes ), # of time stages,
# of lines in figure and # of arrows in figure

READ(4,100) NSTAGE,NSTATE,I,ITR,M NCL=M-I READ(8,100) NLINS,NARW 100 FORMAT(15I3) READ(8,100) (IPRD(I),I=1,NSTATE)

read overall figure scale

READ(8,110) SCX 110 FORMAT(23F6.3) NST1=NSTAGE+1

read tank and process trajectories

I=(ITR+1)*1000 FIND(7'I)
DO 10 I=1,NSTAGE
READ(7,111) (TRAJ(J,I),J=1,NSTATE)
FORMAT(3X,15F7.3)
READ(7,110) (UT(J,I),J=1,M),(EX(J,I),J=1,3)
CONTINUE
READ(7,111) (TRAJ(J,NST1),J=1,NSTATE)
C
I=1000*(11+3*NSTAGE+2*NCL+NSTATE)
FIND(4'I)
DO 11 I=1,NST1
READ(4,110)(CS1(J,I),J=1,NSTATE)
READ(4,110)(CS2(J,I),J=1,NSTATE)
CONTINUE
C******************************************************
**************
C State (tank) trajectories are displayed inside aqua ovals
C******************************************************
C
* 91: ERASE SCREEN;
FIND(8'4000)
DO 20 I=1,NSTATE
READ(8,110) X,Y,YMAX
TMAX(I)=YMAX
I=IPRD(I)
CALL PTC(YMAX,TRAJ,I,NSTAGE)
DISPLAY OVAL COLOUR AQUA SCALE SCX,SCX AT X,Y;
DISPLAY AXIS SCALE SCX,SCX AT X,Y;
DISPLAY LEVL COLOUR YLW SCALE SCX,SCX AT X,Y;
CALL PTC(YMAX,CS1,I,NSTAGE)
DISPLAY LEVL COLOUR BLUE SCALE SCX,SCX AT X,Y;
CALL PTC(YMAX,CS2,I,NSTAGE)
DISPLAY LEVL COLOUR BLUE SCALE SCX,SCX AT X,Y;
CONTINUE
C ******************************************************
**************
C Control (process) trajectories are displayed inside red squares
C******************************************************
C
DO 30 I=1,M
READ(8,110) X,Y,YMAX
CSU(I)=YMAX
CALL PTD(YMAX,UT,I,NSTAGE,20)
DISPLAY SQUARE COLOUR RED SCALE SCX,SCX AT X,Y;
DISPLAY AXIS SCALE SCX,SCX AT X,Y;
DISPLAY LEVL COLOUR YLW SCALE SCX,SCX AT X,Y;
CONTINUE
C
C** Lines are drawn to connect the processes - tanks
C*****************************************************
C
C    DO 40 I=1,NLINS
    READ(8,110) X,Y,X1,Y1
* DISPLAY LINE FROM X,Y TO X1,Y1 COLOUR GRN;
40 CONTINUE
C
C*****************************************************
C    Arrows are added to the end of specified lines
C*****************************************************
C
C    DO 50 I=1,NARW
    READ(8,130) X,Y,X1
130 FORMAT(2F6.3,F7.2)
* DISPLAY ARROW ANGLE X1 DEG SCALE SCX,SCX AT X,Y
COLOUR GRN;
50 CONTINUE
C
C*****************************************************
C    Total steam, T/G, purchased power are displayed if desired
C*****************************************************
C
C    DO 70 I=1,3
    READ(8,110) X,Y,X1
    IF(X1.LE.0.) GO TO 70
    Y2=0.
    DO 60 J=1,NSTAGE
        Y2=AMAX1(Y2,EX(I,J))
60 CONTINUE
    CALL PTD(Y2,EX,I,NSTAGE,3)
* DISPLAY AXIS SCALE X1,X1 AT X,Y;
* DISPLAY LEVL COLOUR YLW SCALE X1,X1 AT X,Y;
70 CONTINUE
C
C*****************************************************
C    Text is added to label the blocks of the mill mode
C*****************************************************
C
C    READ(8,100) NTX
    DO 80 I=1,NTX
READ(8,120) X,Y,L,(I$TEXT(J),J=1,L)
120 FORMAT(2F6.3,I3,6A2)
* TXT : TVALUE(I$TEXT,(2*L)) AT X,Y;
* DISPLAY TXT;
80 CONTINUE
1C
C*****************************************************************************
**************
C Keyboard I/P selects the next figure
C*****************************************************************************
90 WRITE(6,160)
160 FORMAT(/,' ENERGY(E) TANK(T) PROCESS(P) WHOLE(W) S TOP(S) .ENTER PLEASE')
READ(5,150) IKEY
150 FORMAT(A 1)
IF(IKEY.EO.IANS(5)) GO TO 92
IF(IKEY.EO.IANS(4)) GO TO 91
IF(IKEY.EO.IANS(3)) CALL PROC(CSU,NSTAGE,UT,M,)
IF(IKEY.EO.IANS(2)) CALL TANK(TMAX,NSTAGE,TRAJ,CS1
9CS2,IPRD,NSTATE,&90)
94 IF(IKEY.EO.IANS(1)) CALL ENERGY(EX,NSTAGE,&90)
GO TO 90
1C
C*****************************************************************************
**************
C TEK contains function key definitions, it is copied here, because LIG wipes
C out these definitions
C*****************************************************************************
92 CALL MTSCMD('$CO TEK',7)
STOP
END
C
SUBROUTINE PROC(CSU,NSTAGE,UT,M,)*
DIMENSION UT(20,30),CSU(20)
WRITE(6,100)
100 FORMAT(' ENTER PROCESS #')
READ(5,110) I
SUBROUTINE TANK(TMAX,NSTAGE,TRAd,CS1,CS2,IPRD,NSTATE, *)
C*******************************************************
**************
C This routine produces a continuous time plot of th
e
C specified tank level
C*******************************************************
**************
C DIMENSION TRAd(15,31),TMAX(15),IPRD(15),CS1(15,31)
,CS2(15,31)
WRITE(6,100)
100 FORMAT(' ENTER TANK #')
READ(5,110) I
110 FORMAT(10G10.3)
IF((I.LT.1).OR.(I.GT.NSTATE)) RETURN1
YMAX=TMAX(I)
I1=IPRD(I)
CALL PTC(YMAX,TRAd,I1,NSTATE)
* ERASE SCREEN;
CALL YLBL(YMAX)
CALL TIMLB(NSTATE)
* DISPLAY LEVL COLOUR 180.;
* DRAW FROM .21,.90 TO .20,.90;
WRITE(6,120) I
120 FORMAT('!STR /PROCESS #',I3,'/')
RETURN1
END
C
C SUBROUTINE ENERGY(EX,NSTAGE,*)
C******************************************************
C
C        Steam, Turbogenerator and purchased power plots
C******************************************************

DIMENSION EX(3,30), YM(3)

DO 10 I=1,3
    YM(I)=0.
    DO 20 J=1,NSTAGE
        YM(I)=AMAX1(YM(I),EX(I,J))
    20 CONTINUE

10 CONTINUE

C
278.2  *
     ERASE SCREEN;
  279  *
     DISPLAY AXIS SCALE 1.2,.28 AT .85,.870;
     CALL PTD(YM(1),EX,1,NSTAGE,3)
  281  *
     DISPLAY LEVL SCALE 1.2,.28 AT .85,.870 COLOUR 120
  282  *
     DISPLAY AXIS SCALE 1.2,.28 AT .85,.545;
     CALL PTD(YM(2),EX,2,NSTAGE,3)
  283  *
     DISPLAY LEVL SCALE 1.2,.28 AT .85,.545 COLOUR 300
  284  *
     DISPLAY AXIS SCALE 1.2,.28 AT .85,.215;
     CALL PTD(YM(3),EX,3,NSTAGE,3)
  285  *
     DISPLAY LEVL SCALE 1.2,.28 AT .85,.215 COLOUR 180

  288  *
     DRAW FROM .26,.968 TO .25,.968;
     WRITE(6,100) YM(1)
  290 100 FORMAT('!STR/','F6.3','/')
  291  *
     DRAW FROM .26,.643 TO .25,.643;
     WRITE(6,100) YM(2)
  292  *
     DRAW FROM .26,.313 TO .25,.313;
     WRITE(6,100) YM(3)
  294  *
     DRAW FROM .81,.98 TO .8,.98;
     WRITE(6,110)
  296 110 FORMAT('!STR/STEAM/')
  297  *
     DRAW FROM .71,.68 TO .7,.68;
     WRITE(6,120)
  300 120 FORMAT('!STR/TURBOGENERATOR/')
  301  *
     DRAW FROM .7,.34 TO .69,.34;
     WRITE(6,130)
  302 130 FORMAT('!STR/PURCHASED POWER/')
  304 RETURN
  305 END

C
307  *
     SUBROUTINE YLBL(YMAX)
  308 C
309 C******************************************************
C Label the Y axis from 0. to YMAX in 10 increments, and draw
C the horizontal grid lines
C******************************************************************************
C
C
C
313 C
314 DY=YMAX/10.
315 YL=.15
316 Y=0.
317 DO 10 I=1,11
318 *   DRAW FROM .85,YL TO .01,YL;
319 WRITE(6,100) Y
320 100 FORMAT('!STR /',F6.2,'/')
321 Y=Y+DY
322 YL=YL+.07
323 10 CONTINUE
324 RETURN
325 END
326 C
327 SUBROUTINE TIMLB(NSTAGE)
328 C******************************************************************************
329 C
330 C   Draw the vertical grid lines and label the X (time) axis with approx.
331 C   five points
332 C******************************************************************************
333 C
334 DX=.7/FLOAT(NSTAGE)
335 X=.15
336 N1=NSTAGE+1
337 C
338 C******************************************************************************
339 C   draw vertical grid lines
340 C******************************************************************************
341 C
342 DO 10 I=1,N1
343 *   DRAW FROM X,.85 TO X,.12;
344 10 CONTINUE
345 X=.15
346 IDIF=INT(.2/DX)
347 DX=DX*FLOAT(IDIF)
348 C
349 C******************************************************************************
350 C   label time axis
351 C******************************************************************************
SUBROUTINE PTC(YMAX, TRAJ, I, NSTAGE)

DIMENSION TRAJ(15, 31)

RFY = .7/YMAX
RFX = .7/FLOAT(NSTAGE)
RX = .15

DO 10 J = 1, NSTAGE
    RX2 = RX + RFX
    LEVL :- LEVL + LINE FROM RX, (TRAJ(I, J)*RFY+.15)
          TO RX2, (TRAJ(I, J+1)*RFY+.15)

10 CONTINUE
RX=RX2
10 CONTINUE
RETURN
END

SUBROUTINE PTD(YMAX, U, I, NSTAGE, ID1)
C***********************************************************************
**************
C This routine finds the line segments needed for a discrete time plot (process rates)
C***********************************************************************
**************
C
DIMENSION U(ID1,30)
C
RFY=.7/YMAX
RFX=.7/FLOAT(NSTAGE)
RX=.15
* LEVL :- BLANK;
DO 10 J=1,NSTAGE
   RX2=RX+RFX
   R=U(I,J)*RFY+.15
   * LEVL :- LEVL + LINE FROM RX,R TO RX2,R;
   RX=RX2
10 CONTINUE
RETURN
* END :
* EOF EOF EOF