12835

SHORT-TERM THERMAL UNIT COMMITMENT BY DYNAMIC PROGRAMMING APPROACH

A MASTER'S THESIS

in

Electrical and Electronics Engineering Department

Middle East Technical University

12835

T. C.
Yükseköğretim Kurulu
Dokümantasyon Merkezi

by

ERHAN GÖKÇAY

FEB, 1991

Approval of the Graduate School of Natural and Applied Sciences.

for Prof.Dr.Alpay ANKARA
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science in Electrical and Electronics Engineering.

Prof. Dr. Erol KOCAOĞLAN
Chairman of the Department

We certify that we have read this thesis and that in our opinion it is full adequate, in scope and quality as a thesis for the degree of Master of Science in Electrical and Electronics Engineering.

Assoc. Prof. Dr. Nezih GÜVEN
Supervisor

Examining committee in Charge :
Prof. Dr. Ayhan Türeli
Assoc. Prof. Dr. Nevzat Özay
Assoc. Prof. Dr. Osman Sevaioğlu
Prof. Dr. Arif Ertaş
Assoc. Prof. Dr. Nezih Güven

ABSTRACT

SHORT-TERM THERMAL UNIT COMMITMENT BY DYNAMIC PROGRAMMING APPROACH

GÖKÇAY, Erhan

M.Sc. in Electrical and Electronics Engineering

Supervisor : Assoc. Prof. Dr. Nezih Güven

February 1991, 165 pages

The operation of a power system has to be planned beforehand in order to provide the optimum operation with several operational constraints. Unit commitment produces the optimum schedule of generators for a short time of interval, usually days for short term planning operation.

In this thesis the unit commitment problem is solved through a computer simulation using dynamic programming. The solution quality is improved by adding all constraints and by using a new, improved economic dispatch. The schedule obtained takes the unit operating constraints, costs, generation and reserve constraints into account. This procedure is also implemented in a multi-processor environment to speed up the solution algorithm. The results obtained are quite promising for such applications.

Key Words : Unit Commitment, Power Generation Dispatch, Dynamic Programming, Parallel Processing

Science Code : 608.02.01

ÖZET

DİNAMİK PROGRAMLAMA YAKLAŞIMI ILE KISA DÖNEM ÜNİTE TAHSİSİ

GÖKÇAY, Erhan
Yüksek Lisans Tezi, Elektrik ve Elektronik Müh. Bölümü.
Tez Yöneticisi : Doç. Dr. Nezih Güven
Şubat 1991, 165 Sayfa

Modern enerji sistemlerinin, optimal çalışmayı sağlamak üzere planlanması gerekir. Ünite tahsisi kısa dönem plan aşamasında kullanılır ve 1-6 gün için ünitelerin en verimli üretim programını saptar.

Bu tezde ünite tahsisi problemi bir bilgisayar simulasyonu ile dinamik programlama yöntemi kullanılarak çözülmüştür. Güç sisteminin, ünite, üretim ve hazır güç reservi gibi çeşitli kısıtlamaları eklenerek çözüm iyileştirilmiştir. Ayrıca problemin çok-işlemcili bilgisayarlarda hızlandırılması için yapılan simulasyon programından umut verici sonuçlar elde edilmiştir.

Anahtar Kelimeler : Ünite tahsisi, güç üretimi optimizasyonu, dinamik programlama, paralel işleyici
Anabilim Dali Sayısal Kodu :608.02.01

ACKNOWLEDGEMENTS

The author is gratefully indebted to Assoc. Prof. Dr. Nezih Güven for his supervision, guidance and continuous interest.

Special thanks to Ali Göktoğan for his support in providing access to the computer equipment and software.

The author is also grateful to Didem Ünlü for her encouragement throughout the project.

TABLE OF CONTENTS

	Page
ABSTRACT	ii
OZET	iii
ACKNOWLEDGEMENT	iv
LIST OF FIGURES	ix
LIST OF TABLES	x
LIST OF SYMBOLS	xi
1. INTRODUCTION	1
2. UNIT COMMITMENT	5
2.1. Computer Operation of Power Systems	5
2.2. Unit Commitment Problem	8
2.2.1. Unit Costs	10
2.2.1.1. Fuel Cost	11
2.2.1.2. Maintenance Cost	11
2.2.1.3. Start-Up Cost	11
2.2.1.4. Shutdown-Cost	12
2.2.2. Unit Constraints	13
2.2.2.1. Minimum Up/Down Times	13
2.2.2.2. Ramp Rate Constraint	14
2.2.2.3. Generation Limits	15
2.2.2.4. Unit Types	15
2.2.2.5. Changes in Unit Generating Limits and Capacity	17
2.2.2.6. Plant Crew Constraint	17
2.2.3. Operational Constraints	17
2.2.3.1. Generation Requirement	17

2.2.3.2. Spinning and Ready Reserve Requirement 18
2.3. Solution teenniques 21
2.3.1. Dynamic Programming 22
2.3.2. Lagrangian Relaxation 23
2.3.3. Branch and Bound Method 24
2.3.4. Benders Decomposition 24
2.3.5. Linear Mixed-Integer Method 25
3. DYNAMIC PROGRAMMING METHOD APPLIED TO UNIT COMMITMENT 27
3.1. Dynamic Programming 27
3.1.1. Fundamental Concepts 27
3.1.2. The Principle of Optimality 28
3.1.3. Applications of the Principle of Optimality to Decision Making 29
3.1.4. The Optimum Path Problem 31
3.1.4.1. Statement of the Problem 31
3.1.4.2. Derivation of the Functional Equation 31
3.1.4.3. Computational Algorithms 33
3.1.5. Characteristics of Dynamic Programming 34
3.1.5.1. Presence of Constraints 34
3.1.5.2. Comparison of dynamic programmin with direct enumaration 35
3.1.5.3. Curse of Dimensionality 36
3.2. Solution of Unit Commitment with Dynamic Programming 37
3.2.1. General Procedure of Dynamic Approach to

Unit Commitment 37
3.2.2. Dynamic Programming Algorithms 40
3.2.2.1. Backward Dynamic Programming Approach 42
3.2.2.2. Forward Dynamic Programming Approach 42
3.2.3. State Space Reduction 44
3.2.3.1. Priority Order 45
3.2.3.2. Truncated Dynamic Programming 46
3.2.3.3. Search Range 47
3.2.3.4. Multipass Dynamic Programming 49
3.2.3.5. Time Dependent Costs 50
4. IMPLEMETATION OF DYNAMIC PROGRAMMING ALGORITHM 52
4.1. Programming Details 52
4.1.1. Input Data 53
4.1.2. State Eliminatiuon 54
4.1.3. Initial Data 55
4.1.4. Economic Dispatch 56
4.1.5. Back-Track Operation 59
4.2. Reduction of State List 60
4.3. Results of a Sample System 1 60
4.4. Results of a Sample System 2 62
5. Improvement of the Algorithm using Parallel-Computing 63
5.2. Application of Parallel-Processing to Dynamic Programming 64
5.2. Results of the Parallel-Processing Implementation 67

6. CONCLUSION 68

6.1. Discussion 69

REFERENCES 73

APPENDIX-A Input/Output Data of the Sample System 1.... 77

APPENDIX-B Input/Output Data of the Sample System 2 ... 83

APPENDIX-C Explanation of the Data Files 97

APPENDIX-D Explanation of the Subroutines 101

APPENDIX-E Flowchart of Operations Planning 103

APPENDIX-F List of the Program 104

LIST OF FIGURES

Figure		Page
3.1.	Optimal Path for a Multistage Decision Process ...	29
3.2.	Optimal Decision Process	30
3.3.	Unit Commitment Paths	39
3.4.	Unit Commitment Algorithm using Backward Dynamic Programming	42
3.5.	Unit Commitment Algorithm using Forward Dynamic Programming	44
3.6.	Truncated Paths in Dynamic Programming	47
3.7.	Unit Selection List	48
3.8.	State Transition for a Single Unit	51

LIST OF TABLES

Table		Page
3.1.	Comparison of dynamic programming with direct enumaration	36
4.1.	Execution times of the sample system 1	61
4.2.	Execution times of the sample system 2	62
A-1.	Generator data for the sample system 1	77
A-2.	Load, spinning and ready reserve limits of the sample system 1	78
A-3.	Ramp rate values for the sample system 1	79
A-4.	Output of the sample system 1 with up/down times ignored	80
A-5.	Output of the sample system 1 with up/down times considered	80
A-6.	Commitment schedule of system 1 when up/down times are neglected	81
A-7.	Commitment schedule of system 1 when up/down times are considered	82
B-1.	Generator data for sample system 2	83
B-2.	Load, spinning and ready reserve values of system 2	87
B-3.	Output of sample system 2 with all constraints	88
B-4.	Commitment schedule of sample system 2 with all constraints	90
B-5.	Output of system 2 with ramp and reserve constraints removed	91

Table Page

B-6. Commitment schedule of sample system 2 with ramp and reserve constraints removed 93

B-7. Output of system 2 with up/down times and reserve constraints removed 91

B-8. Commitment schedule of sample system 2 with up/down times and reserve constraints removed 93

LIST OF SYMBOLS

FuelCost$_i$(P_i) ..... Cost of unit i generating P_i (MW)

HeatRate$_i$(P^i) ..... Heat rate curve for unit i (MBtu/h)

UnitFuelCost$_i$ ..... Fuel cost of unit i ($/Mbtu)

$P_i(t)$ ..... Generation of unit i at time t (MW)

BaseCost$_i$ ..... Base cost of maintenance cost ($)

IncrementalCost$_i$ .. Incremental cost of unit i ($/MW)

StartUpCost$_i$ ..... Start up cost of unit i ($)

TS$_i$ ..... Turbine start-up energy (MBtu)

D$_i$ ..... Number of hours the unit has been down (h)

AS$_i$ ..... Boiler cool-down coefficient

BS$_i$ ..... Boiler start-up energy (MBtu)

MS$_i$ ..... Start-up maintenance cost ($)

ShutDownCost$_i$ ..... Shutdown cost of unit i ($)

$I_i(t)$ ..... On/Off status of unit i at time t

$R_i(t)$ ..... The number of hours that unit i has been online(+) or offline(-) at t

MUP$_i$ ..... Minimum up time of unit i (h)

MDN$_i$ ..... Minimum down time of unit i (h)

PMIN$_i$ ..... The minimum generation of unit i (MW)

PMAX$_i$ ..... The maximum generation of unit i (MW)

Ramp$_i$ ..... Ramp rate of unit i (MW/h)

GR(t) ..... Generation requirement at time t (MW)

INT(t) ..... The sum of net interchange

SPNMAX$_i$ ..... The maximum spinning reserve allowed on unit i (MW)

REQSPN(t) ..... The system requirement for spinning reserve at time t (MW)

SR1,SR2 ..... Percent spinning reserve requirement

RESMAX$_i$ ..... The maximum ready reserve allowed on unit i (MW)

REQRED(t) ..... Total ready reserve at time t (MW)

RR1,RR2 ..... Percent ready reserve requirement

PFast(T) ..... Portion of ready reserve which can be supplied by peakers

Fcost(K,I) ..... Least total cost to arrive at state (K,I)

PCost(K,I) ..... Production cost for state (K,I)

Scost(K-1,L;K,I) ..... Transition cost from state (K-1,l) to state (K,I)

CHAPTER 1

INTRODUCTION

The optimum operation of a power system is the main goal of todays energy management, because of increasing fuel costs. While trying to minimize the cost, a modern power system has to supply energy to users continuously with a reliable service. In order to attain these objectives, a careful planning and operation of the resources is needed. The planning phase of a power system can be divided into three sections according to the time frame they cover, namely short term planning, medium term planning and long term planning. Unit commitment, which is the subject of this thesis is used in the short term planning phase where the time frame is days. Unit commitment is used to determine the start-up and shut-down schedule of generators to meet the forecasted demand for one day to one week. In solving a unit commitment problem, generally two basic decisions are involved, where these decisions are the unit commitment decision and the economic dispatch decision. The unit commitment decision involves determining which generating units are to be running during each hour of the planning horizon, considering system capacity requirements including reserve, and the constraints on the start-up and shut-down of units. The related economic dispatch decision involves the allocation of system demand and spinning reserve capacity

among the operating units during each specific hour of operation. As these two decisions are interrelated, the unit commitment problem embraces both these decisions, and the objective is to obtain an overall least cost solution for operating the power system over the scheduling horizon. Unit commitment is an important task in the daily planning of operations scheduling, because small savings for one week can result of high savings for a year.

The unit commitment problem is a very large nonlinear mixed-integer programming problem. For a system of 100 units, and a time horizon of one week, there are over 30000 decision variables. At the present time, there is no method that solves the unit commitment problem exactly for large systems.

There are several approaches to solve the unit commitment problem. The methods used can be classified into two groups. In the first group, there are mathematical optimization methods where theoretically the schedules found are the optimum schedules, but they can not be applied for realistic systems without any heuristic techniques introduced. Second group methods are heuristic methods that are used in practice but there is no theoretical proof that the optimum schedule can be found. The mathematical techniques can be summarized as follows : lagrangian relaxation, benders decomposition, branch-and-bound methods, integer programming, and dynamic programming. A brief summary of these methods can be found in

Chapter 2. Most of these methods have several limitations and in general some approximations has to be made in order to use these methods for realistic systems. Dynamic programming is used very widely, however application for large systems is limited because of exponential increase of computer resources with increasing number of generators. Lagrangian relaxation is applied very recently to large systems and quite satisfactory results are obtained. Other methods, as of today, have not gained wide usage.

Application of non-linear programming techniques is limited only to greatly simplified problems. Consideration of the pertinent constraints results in a non-convex and non-differentiable problem which makes difficult non-linear programming solutions. Linear programming methods do not generally present satisfactory solutions due to level of approximations introduced by linearizing the problem.

In this thesis, the unit commitment problem is solved using dynamic programming for a time frame up to 168 hours. Since the quality of economic dispatch affects the commitment schedule, the economic dispatch used in unit commitment problem is also solved with dynamic programming. The economic dispatch used here is a spinning reserve, ramp rate, and generation limits constrained economic dispatch. Such a detailed dispatch increases the quality of the optimum schedule found.

In Chapter 2, the computer operation of a power system will be explained with several phases of the planning operation, and the importance of the unit commitment problem in the planning phase is emphasized. Additionally, the unit commitment problem will be described in detail and several constraints of the problem with their effects to the solution will be discussed through related formulae. Several methods and algorithms will be given briefly with several approximations used in these solutions. Chapter 3 will cover the theoretical background of dynamic programming and derivation of the formulation of the algorithm and will discuss the advantages and disadvantages of the algorithm with several computational methods. Also the application of the dynamic programming algorithm to the unit commitment problem will be discussed with several reductions on constraints and their effects to the solution. Results of two sample systems for verification purposes of the program will be presented in Chapter 4, where the explanation of the program units and the structure of the input data files will be discussed. Chapter 5 is about the application of parallel-processing to dynamic programming to speed-up the algorithm. In Chapter 6 the discussion of the results will be given with suggestions for future work.

CHAPTER 2

UNIT COMMITMENT

2.1 Computer Operation Of Power Systems

A modern power system has to provide several operational objectives during its operation. These objectives can be summarized as follows : continuous quality service with acceptable impact upon the environment and with minimum cost to the user.

A brief explanation of the objectives can be stated as follows: The system should be operated so that the energy provided to the user has acceptable voltage and frequency regulation. The conditions during energy production should not be harmful to the environment and to the user. The service has to be continuous, secure and reliable, which means that the system has to recover its original state upon a contingency and it has enough reserves to supply the changing load. Finally, the system has to supply all the objectives described above, with minimum cost.

In order to supply energy, with the objectives described above, the operation of a power system has to be planned beforehand. There are several levels of this planned operation. The load demand of a power system is cyclic in nature and has several peak levels during several time

periods (weekly peak level, or monthly peak level ...etc). The planned operation has to use resources and facilities, such that the peak demand of each cycle is supplied at minimum cost.

The planning of a power system operation has three sections. These sections are as follows :

1) Operations planning
2) Operations control
3) Operations accounting

Operation planning can be divided into three sections with respect to the time frame, namely : Long term operation planning, medium term operation planning, short term operations planning.

The output of the long term planning is the scheduling of all resources yearly on a monthly basis for statisfying the monthly peak load and this is the input of the medium term planning. The output of the medium term planning is the schedule for supplying the weekly peak load, and this is the input of the short term planning which produces the daily and hourly schedule.

Operations control is used to supply instantaneous load within a time frame of seconds and minutes. Some of the tasks performed in operations control are : automatic generation control, state estimation, on-line load flow,

schedule of transactions, economic dispatch calculations, operating reserve calculations, and load management system.

Operations accounting records every event in the system. This data will be used in future planning operations.

-- Long term operations planning :

The long term planning produces the monthly load forecasting and a maintenance scheduling program, where the monthly load forecasting program estimates the peak load demand of every month. The maintenance scheduling program gives the maintenance schedule of units according to the manufacturer specifications for satisfying the montly load demand roughly. The time horizon is months.

-- Medium term operations planning :

The hydro-thermal coordination and a rough load forecasting is produced in this planning stage. The load forecasting calculates the weekly peak load level, and hydro-thermal coordination program calculates the schedule of hydro units and other units such that the weekly demand is supplied with minimum cost.

-- Short term operations planning :

The short term planning performs a short term load forecasting, a security analysis, and a unit commitment program. The short term load forecasting estimates the hourly demand over one week, and the unit commitment program produces the generating units to be committed during the week, such that the weekly load demand is satisfied with minimum cost. Because there is always an uncertainty in the load forecast,and some unplanned outages of the generating units, the unit commitment program has to be rerun several times during a day. Flow diagram and timing of these operations are presented in Appendix A.

2.2 Unit Commitment Problem

Unit commitment is an important optimization task in the daily operation planning of power systems. Unit commitment in a power system involves determining a start-up and shut-down scheduling of units to be used to meet the forecasted demand, over a future short term (24-168 hours) period. The objective is to obtain an overall least total cost solution for operating the power system over the scheduling horizon. In addition to providing generation to meet the load, scheduled units must also provide reserve margins to meet fluctuations in load or to cover equipment failure, and satisfy several constraints such as generation output limits, minimum up/down times and ramp rate constraint of generators

and plant crew constraints.

The scope of the operations scheduling problem (or unit commitment problem) will vary strongly from one utility to another depending on their mix of units and particular operating constraints.

The economic consequences of operations scheduling are very important. Since fuel cost is a major cost component, reducing the fuel cost by even 0.5 percent can result in savings of millions of dollars per year for large utilities.

The time horizon of operations scheduling depends on a number of factors. Large steam units require that they be scheduled over a period of several days. Many hydro or pumped-hydro reservoirs have daily or weekly cycles. Typically the commitment and generating schedule is output on an hourly basis.

The schedules are actually produced for only several hours. This is because the scheduling requires forecast of many stochastic quantities such as loads, hydro inflows and unit avalaibilities. If the actual values of these quantities differ greatly when compared to the forecasts, then it is economical to resolve the operations scheduling problem.

The combined thermal unit commitment and short-term hydro scheduling problem is often referred as the short-term hydro-thermal scheduling problem, which produces hourly

schedules over a period of several days. Modern energy management systems often include unit commitment and, if appropriate, short-term hydro scheduling programs.

The objective of the unit commitment problem is the minimization of the total unit cost, which consists of :

- Fuel costs
- Maintenance costs
- Startup costs
- Shutdown costs

The problem can be formulated as follows :

Minimize

$$\sum_{t} \sum_{i} \left(FuelCost_i(t) + MaintenanceCost_i(t) + StartUpCost_i(t) + ShutDownCost_i(t) \right) \quad (1)$$

Subject to the constraints, generation and reserve requirements which will be defined in detail in Section 2.2.2.

2.2.1. Unit Costs

Unit cost in unit commitment problem may be divided into two categories. The first being the transitional cost while the other is the production or generation cost. Generally the transitional cost is the cost associated with the starting of a unit, and it may also include shutdown cost. The production cost is the fuel cost required to meet the load, and this depends on the unit loading, heat rate and fuel price. The maintenance cost is also included in the production cost.

2.2.1.1. Fuel Cost

The fuel cost for generating P_i MW for unit i can be represented by the equation :

$$FuelCost_i(P_i) = HeatRate_i(P_i) * UnitFuelCost_i$$

where

$HeatRate_i(P_i)$: Heat Rate curve for the unit (MBTU/h)

$UnitFuelCost_i$: Fuel cost (\$/MBTU) of unit i

P_i : Generation of unit i (MW)

A third order polynomial heat rate curve is choosen, which is given as :

$$HeatRate_i(P_i) = A_0 + A_1*P_i + A_2*P_i^2$$

Several modelling approaches can be used for heat rate curve. Typical representations use polynomial curves (up to fifth order) or represent the incremental heat rate curve as piecewise constant or piecewise linear.

2.2.1.2. Maintenance Cost

A typical representation of the maintenance cost is :

$$MaintenanceCost_i = BaseCost_i + IncrementalCost_i * P_i$$

2.2.1.3. Startup Cost

Usually, some form of startup cost is also considered in the unit commitment problem. A simple practice is to assume a constant cost irrespective of the unit down time. However, in order to provide a more accurate measure of the actual cost

involved, a time dependent startup cost is needed. One would expect the startup cost to be dependent on the temperature of the unit and hence its down time. Since the cooling rate of a unit is approximately exponential, an exponential startup cost curve is generally prefered though other forms of cost curves can be used.

Typical costs, that are included, are startup fuel costs, startup maintenance costs, and turbine startup energy. Startup cost can be written as :

$$StartUpCost_i = TS_i * UnitFuelCost_i + (1 - EXP(-D_i/AS_i)) * BS_i * UnitFuelCost_i + MS_i \quad (2)$$

where

TS_i : turbine startup energy (MBTU)

D_i : number of hours the unit has been down

AS_i : boiler cool-down coefficient

BS_i : boiler startup energy

MS_i : startup maintenance cost

2.2.1.4. Shutdown Cost

Normally in practice, no cost is associated with the shutting down of units, shutdown costs may be included in the computation of total cost for generalization. A constant cost is specified for each unit as shutdown cost, and this cost is independent of the length of time the unit has been running before the shutdown. The shutdown costs are applied in the

hour the unit is taken off line.

ShutDownCost$_i$: ShutDown cost of generator i

2.2.2. Unit Constraints

In addition to achieving minimum production cost, a well coordinated thermal generation schedule must also satify several constraints such as generation output limits, ramp rate constraints and minimum up/down times.

Typically, the generators, on which the start-up restriction is of greatest concern, are the larger, more economic coal units. Once one of these units is started, an economic dispatch assigns as much energy to these sources as possible. As a result, if the constraints were ignored altogether, the final unit commitment solution would show an inapplicable system dispatch and possibly an inapplicable commitment schedule. The unit constraints are briefly explained in the following sections.

2.2.2.1. Minimum Up/Down Times

In the operation of units, engineering considerations or manufacturer specifications normally require that a unit be running for at least a certain amount of time before it is shutdown. Similarly, a minimum down time is imposed on individual units between successive operations. This is necessary because of some thermal and mechanical limitations of large coal units. The minimum up/down times can be

formulated as :

$$I_i(t+1) = 1 \quad \text{if} \quad 0 < R_i(t) < MUP_i \tag{3.1}$$

$$I_i(t+1) = 0 \quad \text{if} \quad 0 < R_i(t) < MDN_i \tag{3.2}$$

where

MUP_i : Minimum up time of unit i

MDN_i : Minimum down time of unit i

$I_i(t)$: On/Off status of unit i at time t (1/0)

$R_i(t)$: The # of hours that unit i has been online(+) or offline(-) at hour t.

2.2.2.2. Ramp Rate Constraint

Large, economic coal units have start-up restrictions. Because of some thermal and mechanical restrictions, coal units can not change their generation levels very fast. The allowed generations in the next hour can be a function of the unit's current generation, the number of hours a unit has been on-line, and whether the unit's generation is increasing or decreasing. Ramp limits are difficult to be considered optimally if one is also going to consider spinning reserve constraints and losses. The approaches to ramp limits in the literature have been limited to the economic dispatch problem where the unit commitments are assumed to be known. So some ramp rate constraints are imposed on these units during economic dispatch. The ramp rate constraint can written as :

if $R_i(t) > 1$ then

$$MAX\{ PMIN_i , P_i(t-1)-Ramp_i \} \leq P_i(t) \leq MIN\{PMAX_i, P_i(t-1)+Ramp_i\} \quad (4)$$

and

if $R_i(t) = 1$ then $P_i(t) = PMIN_i$

where

$PMIN_i$: The minimum MW generation of unit i

$PMAX_i$: The maximum MW generation of unit i

$P_i(t)$: Generation of unit i at time t

$Ramp_i$: Ramp rate in MW/hr of unit i

2.2.2.3. Generation Limits

Each unit has some operational limits determined by economical and/or physical considerations, which define the upper/lower generation limits of a generator. The generation limits can be written as :

$$PMAX_i \leq P_i(t) \leq PMIN_i$$

2.2.2.4. Unit Types

There are several unit types. These units types are described briefly below :

-- Unit Avalailibility

Units which are on forced outages or on maintenance are unavailable for commitment and should be taken out of consideration.

-- Must Run Units

The pre-scheduling units, which must be on-line due to operating reliability and/or economic considerations, are called must-run units and are considered in unit commitment procedure for all time period.

-- Units On Fixed Generation

These are the units which, because of economic reasons, Hhave been pre-scheduled and have their generation specified for certain time period. A unit on fixed generation is automatically a must run unit for the designated period.

-- Cycling Unit

Cycling units are the units that can cycle on and off and are subject to minimum up/down time constraints.

-- Peakers

Peakers are units (usually gas turbines) that can be started up quickly and are typically not subject to minimum up/down time constraints. Many utilities require that peakers, when dispatched, operate at their upper generating limit. These units can be counted in the overall reserve assessment as long as their time to come up to full capacity is taken into account.

2.2.2.5. Changes in Unit Generating Limits and Capacity

Partial outages and component maintenance could reduce the unit generating limits and capacity. Similarly, the restoration of components can reduce the capacity of the unit for some time period.

2.2.2.6. Plant Crew Constraint

Thermal plants may have restrictions due to manpower limits or due to some mechanical and thermal limitations on the number of units that can be started or shutdown in a given hour. In these plants, more than one generator can not be started or stopped at the same time.

2.2.3. Operational Constraints

The generating units must have the combined capacity to meet the customer load and scheduled interchange, the primary constraint, with sufficient operating reserve and regulating margin to allow emergency response in either direction .

2.2.3.1. Generation Requirement

The generation requirement, if we ignore losses and transmission constraints, can be written as

$$\sum_{i} P_i(t) = GR(t) + INT(t)$$

GR(t) : Generation Requirement at time t

INT(t) : the sum of net interchange (in/out) (+/-)

2.2.3.2. Spinning and Ready Reserve Requirements

Most utilities have a spinning reserve requirement which specifies the amount of on-line generation capacity, in addition to what is needed to meet the generation requirement, in order to meet random disturbances to the system such as the forced outage of a unit or unforseen load changes. Spinning reserve requirements may be specified in terms of excess megawatt capacity or some form of reliability measures. Practices in allocating the amount and the distribution of spinning reserve requirements vary from one company to another and from one pool to another.

For on-line thermal units, the spinning reserve of a unit is equal to the unit capacity less the unit generation or a fixed percentage of the capacity whichever is lower.

The amount of spinning reserve is an important factor in the assurance of uninterrupted supply to the customers and so is the the distribution of the spinning reserve relative to the load centers and among the various generating plants in the system. Therefore, many utilities also have a ready reserve requirement which specifies that amount of additional reserve that must be available to the system. The system has a longer time to provide ready reserve than spinning reserve requirement. Offline peakers are usually allowed to contribute to ready reserve.

Spinning reserve must be allocated to obey certain rules. Typical rules specify that reserve must be a given percentage of forecasted peak demand, or that reserve must be capable of making up the loss of the most heavily loaded unit in a given period of time. It is also possible to calculate the spinning reserve requirement as a function of the probability of not having sufficient generation to meet the load.

Also the reserve must be allocated among fast-responding units and slow-responding units. This allows the automatic generation control system to restore frequency and interchange quickly in the event of a generating-unit outage.

Reserves, finally, must be spread around the power system to avoid transmission system limitations. Below are the different types of spinning reserve requirements to be observed in the commitment of the units.

-- Fixed Total Spinning Reserve : This is the minimum total effective spinning reserve in the system and is fixed for the time period of interest. Normally the amount is equal to some percentage of the annual peak load responsibility of the system.

-- Minimum spinning reserve for each area : By specifying a minimum spinning reserve for each area within the system, each area is assured of a definite amount of additional avalaible capacity within the area. Such requirement is needed particularly when transmission capacity is considered.

-- Maximum spinning reserve from each unit : This requirement avoids the allocation of a large amount of spinning reserve to a few units. The specified maximum amount for each unit may be reflected by the unit capacity and its response rate in picking up generation.

-- Multi Area Constraints:

Many utilities and power pools have limits on the power flow between regions of their system or between their system and neighbouring systems. Each region may also have seperate spinning or ready reserve constraints.

Reserve and ready reserve requirements can be formulated as :

Reserve requirement

$$\sum_{i} \text{MIN} \{ PMAX_i - P_i , SPNMAX_i \} \geq REQSPN(t) \qquad (5)$$

where

$$REQSPN(t) = SR1*(GR(t)+INT(t)) + SR2 * (\underset{i \in \text{online units}}{\text{MAX}} \{ PMAXi \})$$

$SPNMAX_i$: The maximum spinning reserve allowed on unit i

REQSPN(t) : The system requirement for spinning reserve at time t

SR1,SR2 : Input data (as percent)

Ready Reserve Requirement:

$$\sum_i \text{MIN}\{PMAX_i - P_i, SPNMAX_i + RESMAX_i\} \geq REQSPN(t) + REQRED(t) - PFAST(t) \quad (6)$$

where

$$REQRED(t) = RR1 * (GR(t)+INT(t)) + RR2 * \left(\max_{i \in \text{online units}} \{PMAX_i\}\right)$$

$RESMAX_i$: Max ready reserve allowed on unit i

REQRED(t) : Total ready reserve allowed at time t

RR1,RR2 : Input data (as percent)

PFAST(t) : The portion of ready reserve which can be supplied by fast start units like combustion during time t.

2.3. Solution Techniques

The unit commitment problem is a very large nonlinear mixed-integer programming problem. Many methods have been developed for solving the unit commitment problem. The optimization methods for unit commitment have been based on a number of approaches including dynamic programming, lagrangian relaxation, integer programming, branch and bound methods, and benders decomposition. These methods combine mathematical programming approaches with certain approximations and assumptions to make the problem tractable. Dynamic programming and lagrangian relaxation have been used extensively to develop production-grade unit commitment programs. These methods are explained in the following sections.

2.3.1. Dynamic Programming

Dynamic programming is a methodical procedure which systematically evaluates a large number of possible decisions in a multi-step problem [24,25]. A subset of possible decisions is associated with each sequential problem step such that a single decision must be made in each step. There is a cost associated with each possible decision and this cost may be affected by the decision made in the preceeding step. Additional costs, termed "transition costs", may be incurred in going from a decision in one problem step to a decision in the following problem step over what is termed a "transition path". The objective is to make a decision in each problem step which minimizes the total cost for all the decisions made.

The dynamic programming procedure consists of two parts:an evaluation of all possible configuration in each hour from the beginning to the end of the problem, and a "back-track" operation from the end of the problem back to the beginning over the optimal path. Additional details can be found in Chapter 3.

2.3.2. Lagrangian Relaxation

Generally, the lagrangian relaxation approache states that the unit commitment problem can be written in terms of 1) a cost function that is the sum of terms each involving a single unit, 2) a set of constraints involving a single unit, and 3) a set of coupling constraints (the generation and reserve requirement), one for each hour in the study period, involving all the units [16,17,18,23]. One can show that an approximate solution to this problem can be obtained by adjoining the coupling constraints onto the cost using Lagrange multipliers. The resulting "relaxed" problem is to minimize the so-called Lagrangian, subject to unit constraints.

To apply the lagrangian relaxation method the following questions has to be answered :1) How do we find the multipliers so that the solution to the relaxed problem is near the optimum, 2) How close to the optimum is the solution, 3) How do we solve the relaxed problem. Duality theory has shown that for nonconvex problems there will typically be a "duality gap" between the cost obtained by solving the relaxed problem and the optimum cost for the original problem. Since the commitment decision variables are discrete, the unit commitment problem is nonconvex. The duality gap, however, has been shown to go zero as the problem size gets bigger.

2.3.3. Branch and Bound methods

Branch-and-bound is a technique to solve a discrete variable problem by solving a sequence of simpler problems derived from the original problem [4,2]. The search is organized via a branch-and-bound tree. The top node on the tree is a relaxation of the problem(typically the integer constraints are replaced by interval constraints) that is relatively easy to solve. The successors of the top node are a set of problems(also relaxations of the original problem) each having a disjoint solution space; where the union of the solution spaces of these successors is the solution space of the top node problem. Each of these nodes is also seperated into a set of problems with disjoint solution spaces having the property that the union of the solution spaces is the solution space of the parent node. This process is repeated until the solution space for the end nodes of the tree consists of an enumaration of all the discrete variables. Since the solution space of the parent node is a relaxation of all the successor nodes, the optimum solution of the parent (minimization problem) must be lower than the solution of any of the successors. Therefore, the optimum solution of the parent is a lower bound on any descendant node. The basic idea of branch-and-bound is that if, at any time in the procedure, one obtains a solution of the problem at a node that is greater than a feasible solution to the original problem, then all feasible solutions corresponding to

descendants of that node are known to be nonoptimal. It, therefore, is unnecessary to evaluate these descandant nodes.

2.3.4. Benders Decomposition

Benders decomposition [3] requires that the unit commitment problem be formulated as a mixed-integer linear programming problem. Benders decomposition is a way to decompose the unit commitment problem into a master problem involving only the discrete commitment variables and a subproblem involving the continuous generation variables. The subproblem corresponds to the economic dispatch problem with a given commitment. The marginal costs(system lambda), for each hour, from the subproblem are used to constrain the allowed commitments in the master problem, which can be decomposed into a sequence of problems each involving a single unit. The master problem provides commitments to the economic dispatch problem, where both problems are solved iteratively until the solution converges.

2.3.5. Linear Mixed-Integer Method

By this method the unit commitment problem is formulated as a linear mixed-integer programming problem and then use standart integer programming algorithms to solve for the commitment schedule [19,22]. In this approach the cost function and constraints are written as linear functions of the generation and commitment decision variables. The commitment variables are modelled as having the values 0 and

1 corresponding to a unit being off or on at a given time. The relaxation corresponding to the top node of the decision tree allows all the commitment variables to take on any value in the interval [0..1]. Each descendant of a node corresponds to restricting some of the commitment decisions to 0 and 1. The problem at any node of the tree is a linear programming problem which can be solved using a modern linear programming package.

CHAPTER 3

DYNAMIC PROGRAMMING METHOD APPLIED TO UNIT COMMITMENT

3.1 Dynamic Programming

In the following sections, the details of dynamic programming method are explained theoretically.

3.1.1. Fundamental Concepts

In optimal control problems the objective is to determine a function that minimizes a specified functional, which is the performance measure. The analogous problem in calculus is to determine a point that yields the minimum value of a function. In order to make a brief review on functionals the following definitions are given :

DEFINITION 1 : A function f is a rule of correspondance that assigns to each element q in a certain set D a unique element in a set R. D is called the domain of f and R is the range.

EXAMPLE : Suppose $q_1, q_2, \ldots q_n$ are the coordinates of a point in n-dimensional Euclidean space and

$$f(q) = \sqrt{(q_1^2 + q_2^2 + ... + q_n^2)} .$$

The real number assigned by f is the distance of the point q from the origin.

The definition of a functional parallels that of a function.

DEFINITION 2 : A functional J is a rule of correspondance that assigns to each function x in a certain class @ a unique real number. @ is called the domain of the functional, and the set of real numbers associated with the functions in @ is called the range of the functional.

Notice that a domain of a functional is a class of functions; intuitively one can say that a functional is a "function of a function".

EXAMPLE : Suppose that x is a continuous function of t defined in the interval $[t_0, t_f]$ and

$$J(x) = \int_{t_0}^{t_f} (x(t)\, dt)$$

the real number assigned by the functional J is the area under x(t) curve.

3.1.2. The Principle of Optimality

The optimal path for a multistage decision process is shown in Figure 3.1 . Suppose that the first decision (made at a) results in segment a-b with cost Jab and that the remaining decisions yield segment b-e at a cost of Jbe. The minimum cost Jae* from a to e is therefore :

$$J_{ae}^* = J_{ab} + J_{be}$$

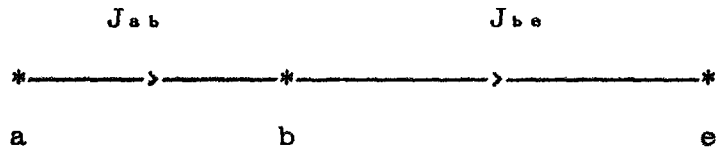


Figure 3.1 Optimal path for a multistage decision process

ASSERTION: if a-b-e is the optimal path from a to e, then b-e is the optimal path from b to e.(proof is straigtforward).

The above property of an optimal policy is called the principle of optimality which states that :

-- An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision [24].

3.1.3. Application of the Principle of Optimality to Decision Making

The following example illustrates the procedure for making a single optimal decision with the aid of the principle of optimality.

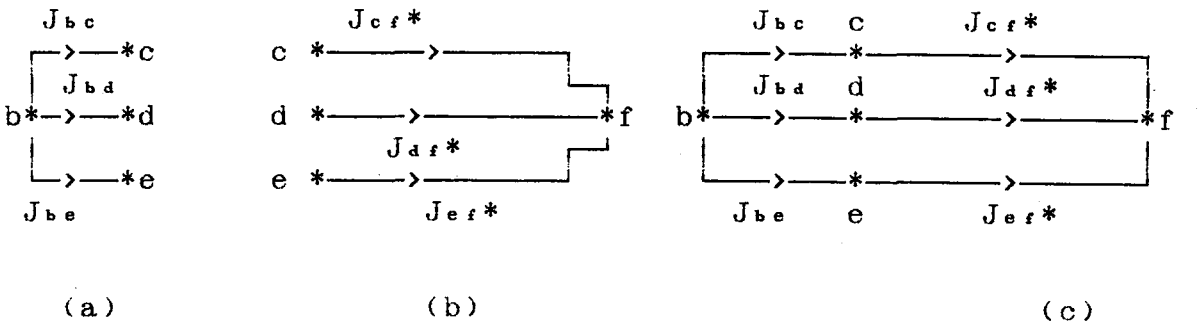


Figure 3.2 Optimal decision process

Consider a process whose current state is b. The paths resulting from all allowable decisions at b are shown in Figure 3.2 (a). The optimal paths from c,d and e to terminal point f are shown in Figure 3.2 (b). The principle of optimality implies that if b-c is the initial segment of the optimal path from b to f, then c-f is the terminal segment of this optimal path. The same reasoning applied to initial segments b-d and b-e indicates that the paths in Figure 3.2 (c) are the only candidates for the optimal trajectory from b to f. The optimal trajectory that starts at b is found by comparing

$$Cost_{bcf}^* = J_{bc} + J_{cf}^*$$

$$Cost_{bdf}^* = J_{bd} + Jd_{df}$$

$$Cost_{bef}^* = J_{be} + J_{ef}^*$$

The minimum cost of these costs must be the one associated with the optimal decision at point b.

Dynamic programming is a computational technique which extends the above decision-making concept to sequences of

decisions which together define an optimal policy and trajectory.

3.1.4. The Optimal Path Problem

The purpose of this section is to illustrate the combined use of the functional equation approach and the "Principle of Optimality" to analyze sequential decision processes and to mention the three main computational algoritms used in general for dynamic programming situations.

3.1.4.1. Statement of the Problem

Consider a set of points $\{P_i\}, i=1,2....N$, with an associated cost matrix $[c_{ij}]$, where c_{ij} is the cost of moving directly from P_i to P_j in one move, and it is assumed that $0 \leqslant c_{ij} \leqslant \infty$. The problem is to find 1) the minimal cost of moving from any point P_i to point P_N using as many moves as required, 2) the optimal sequence of moves.

3.1.4.2. Derivation of the Functional Equation

Define a decision rule or a policy, which is equivalent to some function s(i) which states: if at point i, move to next point j,where j=s(i). The cost of this movement c_{ij}, and, since this depends only on i and j, it is determined completely by the starting point i and the policy s(i), which prescribes j. If generalized, the cost of getting from i to N is determined completely by the given policy and depends, for a fixed N, only on i. The associated cost is denoted by f(i).

During the move from i to j, a direct cost $c_{i,j}$ will be incurred. Once point j is reached, the objective is to move to N in an optimal manner, and by definition this is equal to f(j).

Thus the total cost of getting from i to N, first via j and then optimally to N is :

$$c_{i,j} + f(j)$$

Since j can now be choosen to be best, f(i) satisfies the functional equation

$$f(i) = MIN_i [c_{i,j} + f(j)]$$

This equation is termed a forward functional equation. It is termed forward since the point to move to in the next step is determined at the current state.

Another form of the functional equation is the backward functional equation, which finds the optimal policy from some fixed point, say 1, to any final point i. The functional equation becomes :

$$f(i) = MIN_j [f(j) + c_{j,i}]$$

where f(i) is minimal cost of getting from 1 to i.

3.1.4.3. Computational Algorithms

There are three main methods used in dynamic programming calculations. These methods can be explained as follows :

-- Method of Successive Approximations:

Define a function $f_1(i)$ as follows:

$$f_1(i) = c_{iN}$$

Define a sequence of functions $\{ f_n(i) \}$ inductively as follows:

$$f_n(i) = MIN_j [c_{ij} + f_{n-1}(j)] \qquad (7)$$

Physically, $f_2(i)$ is equal to the minimal cost of getting from i to N in at most two steps, and $f_n(i)$ is equal to the minimal cost of getting from i to N using at most n steps.

It can be shown that the sequence $\{ f_n(i) \}$ converges monotonically to a unique solution of equation (7) among all those solutions with f(N) = 0. Convergence is guaranteed in (N-1) steps [25].

-- Method of Approximations in Policy Space:

In this method an initial policy, $s_0(i)$ is chosen. Then the set of simultaneous equations can be solved where the equation is given as follows :

$$f_0(i) = c_{i,s_0(i)} + f_0(s_0(i)), \quad i=1,2... N$$

and

$$f(N) = 0$$

When $\{f_0(i)\}$ is determined, a policy $s1(i)$ can be found which minimizes for all i the equation below :

$$c(i,s_1(i)) + f_0(s_1(i))$$

The policy $s_1(i)$ is replaced with $s_0(i)$ and the cycle continually repeated until a stationary policy is obtained [25].

-- Direct Method for Directed Sets:

In some cases the set of points form a directed set in the sense that movements from one point can only be in the direction of the ordering of the points. In this case the most efficient computational algorithm is given directly by equation 3.1 and can be summarized as : Begin with f(N)=0, and compute f(N-1), then f(N-2)... and, finally, f(1) [25].

3.1.5. Characteristics of Dynamic Programming

Since a direct search is used to solve the functional equation, the solution obtained is the absolute minimum. Dynamic programming makes the direct search feasible because instead of searching among the set of all admissible values that cause admisible trajectories, we consider only those values that satisfy the principle of optimality.

3.1.5.1. Presence of Constraints:

The presence of constraints simplifies the numerical procedure. For example, if the control is a scalar and is

constrained by the relationship

$$-1.0 < U(t) < 1.0$$

then in the direct search procedure we need to try only values of u in the allowed interval instead of all values of u.

$$-\infty \leqslant U(t) \leqslant \infty$$

3.1.5.2. Comparison of Dynamic Programming with Direct Enumaration

Dynamic programming uses the principle of optimality to dramatically reduce the number of calculations required to determine the optimal path or value. In order to appreciate the importance of the principle of optimality, a comparison of the dynamic programming algorithm with direct enumeration is given below in Table 3.1.

Assume that the admissible state values are quantized into ten levels, and the admissible control values into four levels. The important point is that the number of calculations required by direct enumaration increases exponentialy with the number of stages, while the computational requirements of dynamic programming increases linearly.

Number of stages in the process N	Number of calculations required by dynamic programming	Number of calculations required by direct enumeration
1	40	40
2	80	200
3	120	840
4	160	3400
5	200	13640
6	240	54600
L	40*L	$\sum_{k=1}^{L} [10*4^k]$

Table 3.1 Comparison of dynamic programming with direct enumaration

3.1.5.3. The Curse of Dimensionality

One serious drawback of dynamic programming is that for high dimensional systems the number of high-speed storage locations increases very rapidly. This difficulty is called the "curse of dimensionality" [24]. To avoid this, low-speed storage can be used; however this will dramatically increase the computation time.

3.2 Solution of Unit Commitment with Dynamic Programming

3.2.1. General Procedure of Dynamic Programming Applied to Unit Commitment :

In unit commitment, each possible decision involves the commitment of a different group of generating units. Each group is a subset of all the available generating units and is termed a "configuration". Each problem step is a unit of time, normally one hour [5,6,7,8,9,10,11,12,13,14,20,21].

The evaluation part of the procedure begins at the first hour of the study and moves on to each subsequent hour in sequence. This part of the procedure establishes the transition paths which allow the optimal solution to be identified. At each hour a cost is computed for each possible configuration by making an economic dispatch. Each possible configuration in that hour is then associated with a single configuration in the previous hour. This association of configurations is termed a "link back" procedure since possible configurations in each hour are linked-back-to possible configurations in the previous hour. The link back procedure determines the transition paths between the possible configurations with minimum cost. Each possible configuration has a unique link back path and each link back path connects its associated configuration to a single configuration in the preceeding hour.

In each hour a total cost is computed for every possible configuration in that hour. This total cost is the sum of all the configuration (dispatch) costs and transition (start-up) costs connected by the associated link back path. The configuration in the previous hour is selected such that the total cost for the configuration in the current hour is minimum. This accumulated cost includes all associated commitment costs in the previous hours of the study. The main importance of this representation is that the total cost for a given hour can be expressed in terms of the total costs from the previous hour, but the costs of the remaining hours are not needed.

The back-track operation is the second part of the dynamic programming procedure, which is performed after the evaluation part is completed. The result of the evaluation part is a total cost for every possible configuration in the final hour of the study. Each of these final hour total costs represents a possible commitment profile over the entire study. The commitment profile is the one that will yield the minimum total cost given the associated final hour configuration. The optimal solution of the unit commitment problem is the final hour configuration with the lowest total cost. The optimal configuration at each hour is found by tracing over the link back path starting from the end, up to the beginning. The link back path which is traced over is the one associated with the final hour configuration that has the

lowest total cost.

The total cost for a given hour can be expressed in terms of the total costs from the previos hour. For example the costs at time 3 in Figure 3.3 can be written as :

$$\text{Cost}\left(\text{total}_{D1}\right) = \text{Cost}(\text{total}_{C1}) + \text{Cost}(C1\text{-}D1) + \text{Cost}(D1)$$

$$\text{Cost}\left(\text{total}_{D2}\right) = \text{Cost}(\text{total}_{C1}) + \text{Cost}(C1\text{-}D2) + \text{Cost}(D2)$$

$$\text{Cost}\left(\text{total}_{D3}\right) = \text{Cost}(\text{total}_{C2}) + \text{Cost}(C2\text{-}D3) + \text{Cost}(D3)$$

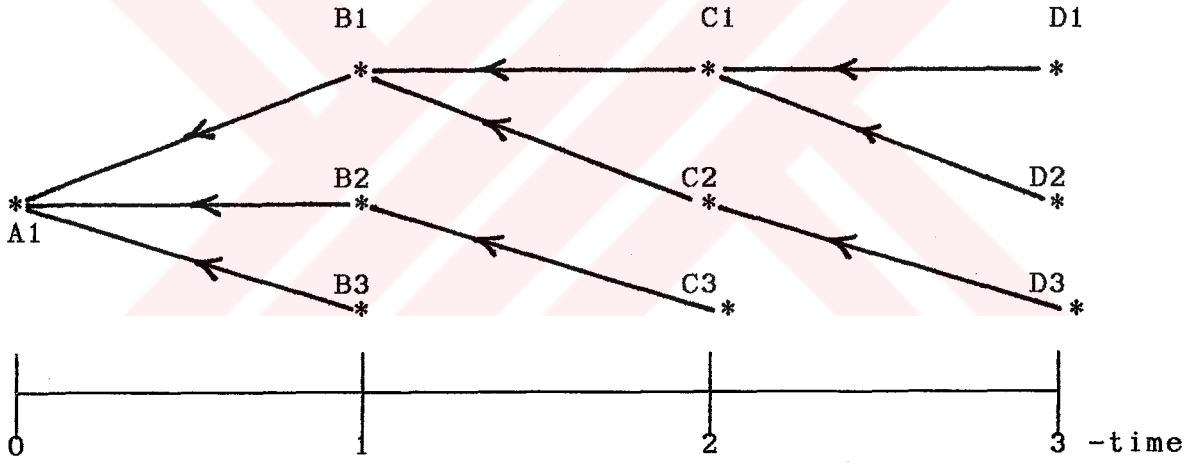


Figure 3.3 Unit commitment paths

A given possible configuration in a given hour can be linked back to any possible configuration in the preceeding hour and a different cost will result for each possible link back. For example there are three different link-back paths and hence three different costs for decision D2, shown in Figure 3.3 with dashed lines. These costs can be written as :

$$\text{Cost}(\text{total}_{D2})a = \text{Cost}(\text{total}_{C1}) + \text{Cost}(C1\text{-}D1) + \text{Cost}(D2)a \quad (8.1)$$

$$\text{Cost}(\text{total}_{D2})b = \text{Cost}(\text{total}_{C2}) + \text{Cost}(C2\text{-}D2) + \text{Cost}(D2)b \quad (8.2)$$

$$\text{Cost}(\text{total}_{D2})c = \text{Cost}(\text{total}_{C3}) + \text{Cost}(C3\text{-}D3) + \text{Cost}(D2)c \quad (8.3)$$

The dynamic programming states that the cost for a configuration must be minimum. So the link-back for a possible configuration is chosen such that the total cost is minimum for that configuration. In Figure 3.3 it was assumed that cost B.1 was the lowest among the other costs B.2 and B.3. So according to dynamic programming approach D2 have to be back-linked to configuration C1. This procedure indicates that when the last hour is reached the optimum configuration is found.

3.2.2. Dynamic Programming Algorithms

There are two approaches to dynamic programming solution of unit commitment : Backward dynamic programming and forward dynamic programming, where these approaches are referred as backward and forward according to time in this application.

3.2.2.1. Backward Dynamic Programming Approach

The first method in dynamic programming uses a backward approach in which the solution starts at the last interval and proceeds back to the initial point. There are M intervals in the period to be considered. The dynamic programming equations to calculate the optimum schedule or path during a time period K, are given as:

$$Fcost(K,I) = \min_{\{J\}} [\, Pcost(K,I) + Scost(I,K;J,K+1) + Fcost(K+1,J) \,] \quad (9)$$

$$Fcost(M,I) = Pcost(M,I)$$

where

Fcost(K,I) : Minimum total fuel cost from state I in interval K to the last interval M.

Pcost(K,I) : Minimum generation cost in supplying the load during interval K given state I.

Scost(I,K;j,K+1) : Imcremental start-up cost going from state I in the K^{th} interval to state J in the $(K+1)^{th}$ interval.

{J} : Set of feasible states in interval K+1.

The production cost Pcost(K,I) is obtained by economically dispatching the on-line units in state I. A path is a schedule starting from a state in interval K to the final interval M. An optimal path is one for which the total cost is minimum. The procedure is shown by the flowchart in Figure 3.4 .

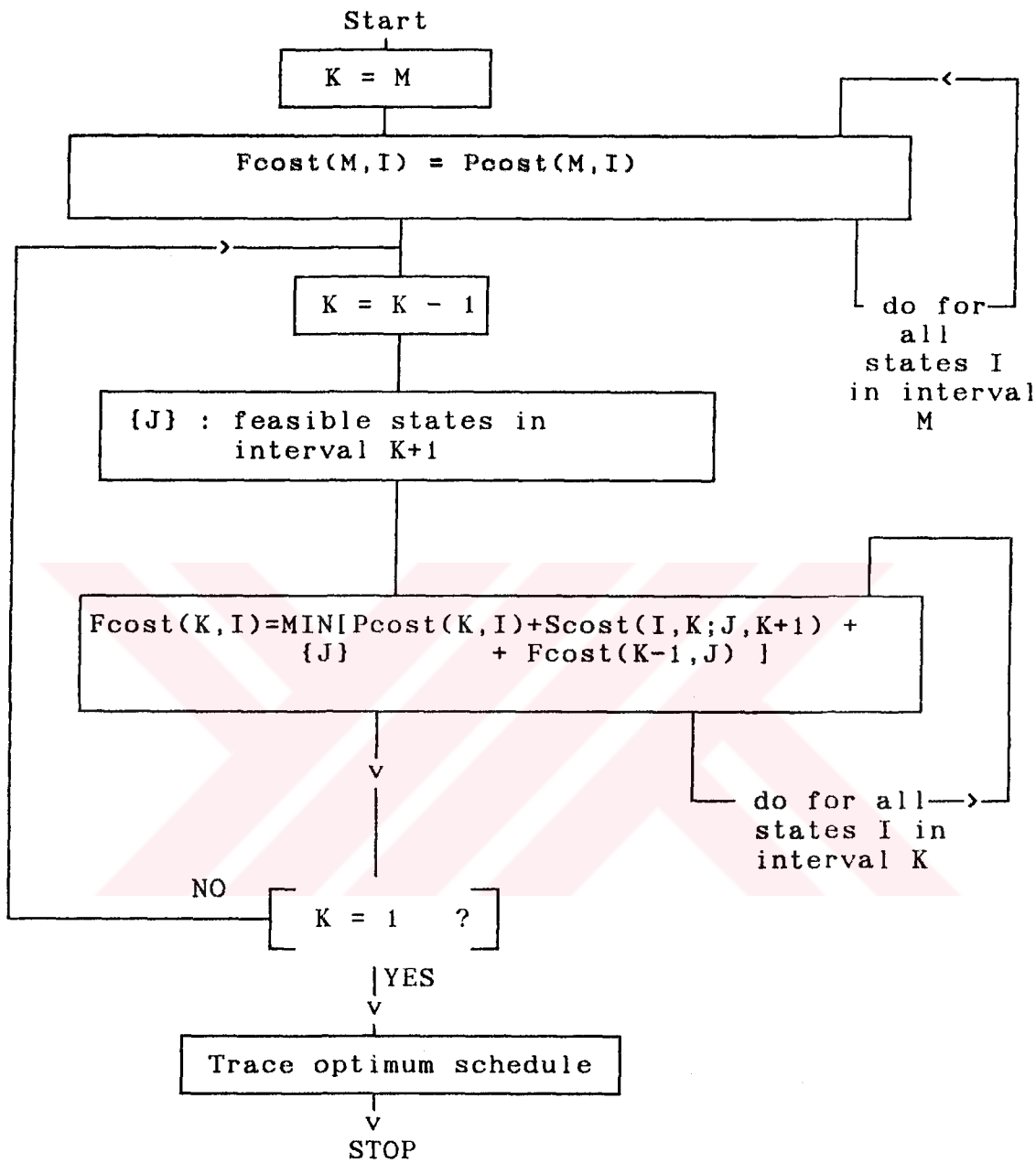


Figure 3.4 Unit commitment algorithm using backward DP.

3.2.2.2. Forward Dynamic Programming

Many practical cases can not be covered with the backward dynamic programming, especially when one considers time dependent unit costs, such as start-up costs, where it is a

function of the time the unit has been off-line. The forward dynamic programming algorithm is more suitable since the previous history can be computed at each stage. Other practical reasons to use forward dynamic programming algorithm is as follows: 1) The initial conditions can be easily specified and 2) computation can go forward in time as long as computer storage is avalaible.

The recursive algorithm to compute the minimum cost in hour K with combination I is :

$$Fcost(K,I) = \min_{\{L\}} [\, Pcost(K,I) + Scost(K-1,L:K,I) + Fcost(K-1,L) \,] \qquad (10)$$

where

Fcost(K,I) = least total cost to arrive at state (K,I)

Pcost(K,I) = production cost for state (K,I)

Scost(K-1,L:K,I) = transition cost from state (K-1,L) to state (K,I)

where state (K,I) is the I^{th} combination in hour K.

{L} = Set of feasible states in interval K-1

For the forward dynamic programming approach, strategy is defined as the transition, or path, from one state at a given hour to a state at the next hour.

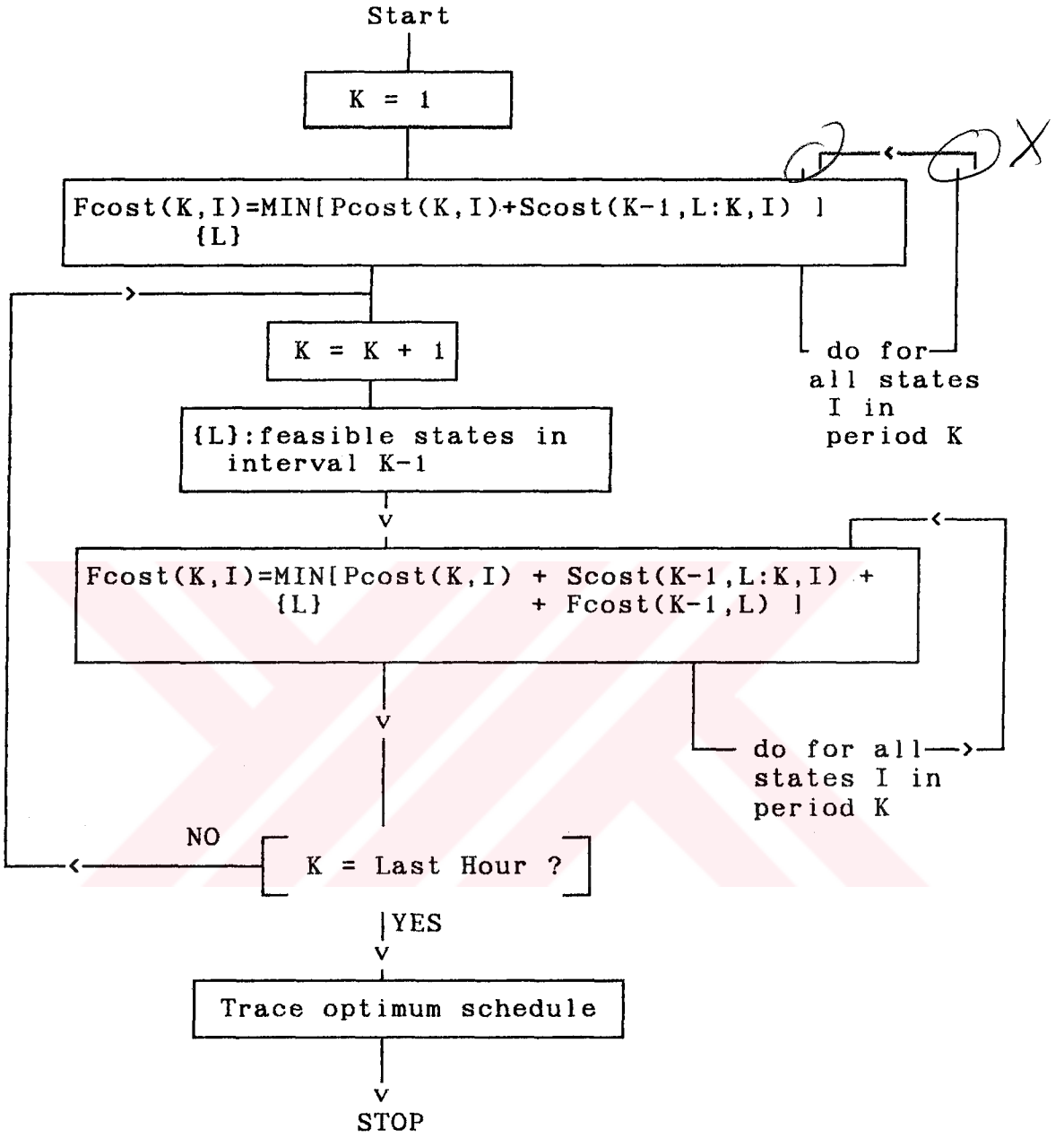


Figure 3.5 Unit commitment algorithm using forward DP.

3.2.3. State Space Reduction

Because of "curse of dimensionality" of dynamic programming the state space, which is defined as the number of different configurations at a given time, can be quite

large. For a system having 15 cycling units, the size of the state space is 2^{15} which is 32768. Calculation of this many dispatches for one week would use large amount of computer memory and computer execution time. In its basic form the dynamic programming procedure evaluates every possible configuration in every hour. Some of these states or configurations are infeasible meaning either the total capacity is not sufficient to supply load and reserve requirements or the total minimum capacity exceeds load. Such infeasible states can be discarded immediately. However for typical systems, the number of states is still very large. For large utilities, the size of memory required and computational requirements, especially CPU time, will exceed the capabilities of even the largest computer.

For these reasons, implementations of dynamic programming require certain approximations which reduce the state space. Some approximation methods are summarized below :

3.2.3.1. Priority Order

Assume the units will be committed and decommitted in a certain priority order. This priority order is a function of the units' operating and startup cost. If there are 10 cycling units, then the maximum state space would be 11 which is much smaller than $2^{10} = 1024$. This method will cause some loss of optimality since the optimal use of the units may depend on the relative size of the units and on the effect of

start-up cost. It may be more economical to commit a generator which is expensive for a short time, for example a peaker, than to commit a large unit for all time, for example a coal unit.

3.2.3.2. Truncated Dynamic Programming

One can introduce two more variables, X and N into the dynamic programming algorithm in order to control the computational efforts. At each hour define a variable X which is the number of states to search, and a variable N which is the number of strategies, or paths to be saved at each hour. For complete enumaration of states, the upper bound of X or N is 2^n-1, whereas n is the number of generators. For strict priority ordering the upper bound is n. Reducing the number N means that the procedure discards the highest cost intervals at each time step and saves only the lowest N paths. The number X further restricts the search range by limiting the number of states to be searched. There is no proof that the theoretical optimal schedule will be found using these reductions. Some algorithms use variable truncated dynamic programming where the search range is increased during the high load changes. An example on truncated dynamic programming is given in Figure 3.6 .

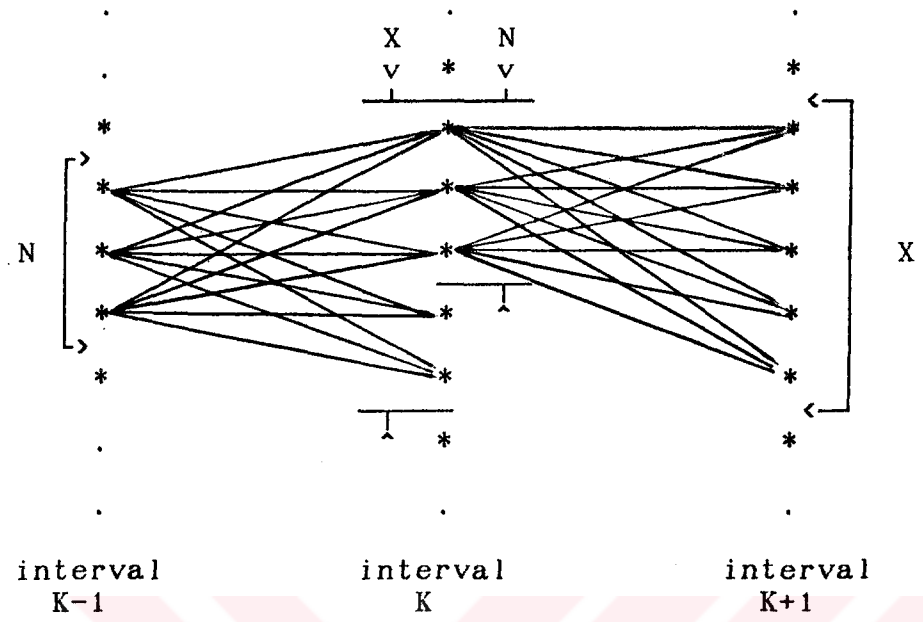


Figure 3.6 Truncated paths in dynamic programming with N=3 and X=5.

3.2.3.3. Search Range

Suppose a system with 13 units where 4 of them must be online at a given hour. These must-run units will form the first four units in the selection list. Must-out or unavailable units are deleted, and the remaining units are added onto the selection list with a priority order. Then the minimum number of units which must be on-line in the order of the selection list so as to meet the load plus total system spinning reserve requirements is determined. Assume that this number is determined as 8. Then a search is performed around unit 8 such that some units above and below these units may be included. This search range is specified with two numbers, one is the number of units below unit 8 and the other number

is the number of units in the search range (e.g. 1 and 4 respectively). Then a search range from unit 7 to unit 10 is specified. Units 5 and 6 become must-run units. Units beyond 10 are called excess units.

For this example the search range has 4 units, giving 2^4-1 = 15 combinations. For those combinations, that does not satisfy load plus reserve requirements units from the excess list are added in their priority order. If the addition does not satisfy the load this combination is discarded. The idea is illustrated in Figure 3.7 .

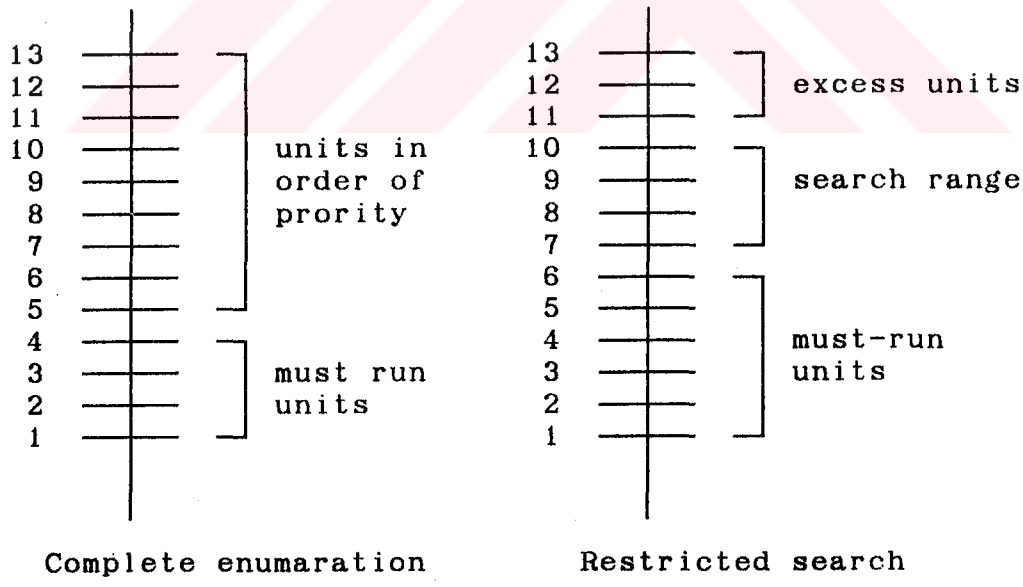


Figure 3.7 Unit selection list.

3.2.3.4. Multipass Dynamic Programming

The multipass dynamic programming is an iterative procedure beginning with a coarse time and state grid and refining the grid pass by pass [14]. In the first pass a very coarse optimum policy is found. The optimal trajectory of the first pass is then used as the nominal trajectory for the second pass. In the second pass a finer decision process is used to find the optimal policy. Admissible state values during this pass include values on the nominal trajectory plus the states with finer units above and below the nominal values. The coarseness of the time and state increments is thus reduced.

In a similar way, the optimal trajectory of each pass becomes the nominal trajectory of the next succeeding pass with the coarseness of the time and state increments being reduced each time. Finally, an increment of one state is used in a certain pass.

Once the pass with one state increment is reached, the time and state increment will not be refined any longer. But additional passes are provided, because the optimal trajectory of final pass may lie along a boundary of admissible states for some time stages. The process continues until the optimal trajectory of the pass remains unchanged.

The process is iterative and hence it may converge to a local optimum instead of a global optimum.

3.2.3.5. Time Dependent Costs

In order to include the start-up costs and the minimum up/down constraints in an optimal manner, the accumulated up/down time of the units have to be included in the state space. Therefore, instead of a state space of 2^{10} for the 10 unit cases, the state space becomes M^{10} where M is related to the sum of the minimum up/down times and the time constant in the startup cost calculation (typically M = 10). Therefore, also with a reduced state space, including the up/down times into the state space is impractical. Instead, an approximate forward DP approach is used where the cumulative up/down time is calculated. This up/down time corresponds to the optimal trajectory to this state. The cumulative up/down time is calculated which is based on the cumulative up/down time of the previous state. If the cumulative up/down time does not violate any up/down time constraint, it is accepted. It is important that this reduction of state space is an approximate procedure and will yield a suboptimal solution. An example is given in Figure 3.8 to show how up/down time can be added to state list.

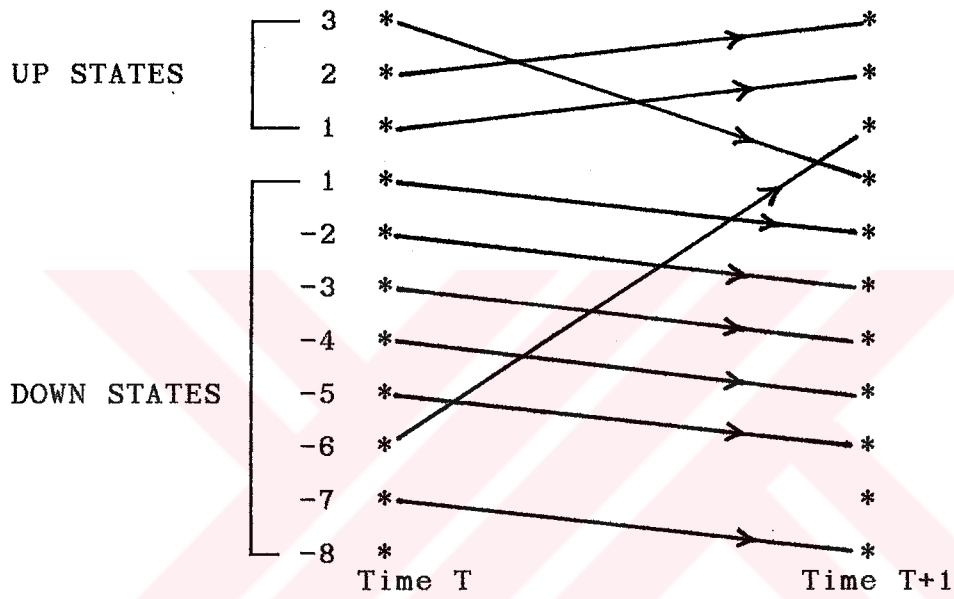


Figure 3.8 State transitions for a single unit.

CHAPTER 4

IMPLEMENTATION OF DYNAMIC PROGRAMMING ALGORITHM

The forward dynamic programming approach, which is described in Chapter 3 has been implemented using the formulations in Chapter 2 with VAX-Fortran programming language on a VAX 8250, and the same program is written also using Turbo-Pascal ver. 5.0 on an IBM PC/AT compatible personal computer in order to have the advantage to run the program on different operating systems. A detailed explanation and list of the subroutines and functions used can be found in Appendix D. The list of the programscan be found in Appendix F.

4.1. Programming Details

A pseudo-code of the program can be given as :

```
        PROGRAM UNIT COMMITMENT
read input data
eliminate non-feasible states from state list
initialize first hour data
for hour = 1 to MaxHour do
  for state = 1 to MaxState do
    for prevstate = 1 , Maxprevstate
      calculate total cost of a configuration for a
      given state and hour and save the minimum cost and
      state from the previous state.
```

```
    endfor
  endfor
endfor
back-track the optimum path
print the results
end
```

4.1.1. Input Data

The input data is entered to the program via data files. Some data files have to be initialized before the program is executed as a necessity. There are several data files, of which the detailed syntax is ilustrated in Appendix C. The data files can be listed as follows :

LOAD.DAT : Data file for the hourly demand list.

INTCHAN.DAT : Data file for hourly interchange loads.

SR12.DAT : Data file for percentages used in spinning reserve calculations

RR12.DAT : Data file for percentages used in ready reserve calculation.

AREA.DAT : Data file for storing area reserve limits.

PLANT.DAT : Data file for storing plant crew constraints.

GEN.DAT : Data file for storing all data of generators used in the program. Additionally this data file includes all the generator constraints

Most of these data files have input data on a hourly

Most of these data files have input data on a hourly basis. Most of the constraints can be edited and changed very easily, so that the adaptation of the system parameters to another system can be easily done. In order to disable a constraint, only a zero value or a high value over all practical limits is entered.

The generator data are assumed to be constant in time, where also the implementation of the opposite case, which means time dependent generator data, can be done very easily. In this program, the only time dependent generator data is generator's up/down ramp rates. Time dependent ramp rates are essential for large coal units, since their operation limits are dependent on temperature and therefore to the time the unit has been on-line. Also up and down ramp rates of these units can be different for a particular time.

4.1.2. State Elimination

Before the calculations proceed, a presearch is made on the state list of generators, which eliminates obvious infeasible states from the state list. The infeasible states are the states, where the total maximum generation at a particular time is lower than the demand plus net interchange, or the minimum total generation capacity of generators is higher than the demand plus net interchange.

In this presearch, the unavailable units are discarded completely. The must-run units and fixed-MW units are given

the ON status during the problem period. These are calculated automatically during a dispatch for a particular combination.

The total generation capacity of peakers is summed up, which is used in calculating the ready reserve requirement. Also, combinations, for which the total maximum reserve contribution does not give enough spinning and ready reserve margins, are eliminated from the state list.

This elimination is important because of the high dimensionality of the problem for number of high generator. For high number of generators further reductions have to made in order to have a reasonable computation time.

Also the time dependent costs are not added to the state list, but an approximate solution is obtained using the reduction technique explained in Chapter 3.2.3.5.

4.1.3. Initial Data

Initial data of the generators is important, because if the initial data are not given, the commitment will be different when ramp rates of some generators are considered. Therefore the initial data can change the commitment schedule. The best initial data are the status and outputs of generators prior to the start time of the unit commitment algorithm.

4.1.4. Economic Dispatch

The economic dispatch algorithm in the unit commitment problem is also solved with dynamic programming. Solving economic dispatch with dynamic programming requires more computational time than other methods (e.g., lagrangian relaxation), but the solution quality of dynamic programming is better since it handles constraints easily. Another reason for using dynamic programming is to find a good optimum in economic dispatch that affects the solution of unit commitment problem. The economic dispatch is chosen to be a constrained one, because omitting the constraints from economic dispatch may result in the elimination of many feasible states which can be in the solution space.

In order to use dynamic programming in economic dispatch problem the following conditions have to be satisfied. The dispatched units have to be ordered, however, for this particular problem changing this order does not affect the results. Additionaly, the demand and generator outputs have to be discretized.

Define the following variables :

P_i = loading on unit i (MW)

$F_i(P_i)$ = cost for generating P_i MW on unit i ($/h)

$f_i(D)$ = stage cost of supplying a demand of D MW with i units

At each stage the cost is calculated using the recursive

formula, which is given as follows :

$$f_i(D) = \underset{\{P_i\}}{\text{MIN}} [\, f_{i-1}(D-P_i) + F_i(P_i) \,]$$

The algorithm works as follows : First the cost of supplying demand for each discrete demand level with one generator is calculated. In the second stage the cost of supplying demand with two generators is calculated, such that the operation of two generators is optimum. And the above procedure is repeated for the other generators also, until all of them are processed. At the last step of the algorithm the minimum cost and operation of all generators to supply the demand is found. A brief explanation of the procedure is given as :

-- at the first stage

$$f_1(D) = F_1(D)$$

-- at the second stage

$$f_2(D) = \underset{\{P_2\}}{\text{MIN}} [\, f_1(D-P_2) + F_2(P_2) \,]$$

where

$f_1(D-P_2)$ is the cost of supplying the remaining portion of demand D, after P_2 is subtracted, and $F_2(P_2)$ is the cost of supplying P_2 with generator 2. The sum of these two quantities is the total cost of supplying demand D with two generator and P_2 is choosen such that it minimizes the cost. This calculation is done for each discrete value of D.

-- at the third stage

$$f_3(D) = \underset{\{P_3\}}{MIN} [\, f_2(D-P_3) + F_3(P_3) \,]$$

where

$f_2(D-P_3)$ is the cost of supplying the remaining portion of demand D, after P_3 is subtracted, with the two generators calculated at the second stage and $F_3(P_3)$ is the cost of supplying P_3 with generator 3. The sum of these two quantities is the total cost of supplying demand D with three generators and P_3 is choosen such that it minimizes the cost. This calculation is done for each discrete value of D.

This calculation continues until the last generator is reached and at the last step the minimum cost of supplying the demand D is found.

In order to decrease the computation time, a multipass dynamic programming method is used. First a coarse optimum is found and the algorithm is repeated in smaller steps in the range of the coarse optimum. This will work correctly if the functions are smooth and convex in the neighborhood of the coarse optimum. If not, the entire range has to be covered with a finer grid. Both the multi-pass and finer-grid methods are implemeted in the program.

One important feature of the economic dispatch solution with dynamic programming is that no restrictions are imposed on the individual unit cost characteristics. They do not have to be convex, smooth, or even possess derivatives. Of course

in the case of nonconvex characteristics the multi-pass method can not be used.

The economic dispatch algorithm considers the following constraints :

- Plant crew constraint
- Reserve and ready reserve limits
- MultiArea reserve and ready reserve limits
- Generation limits of the generators
- Ramp rate limits of the generators

Infinite costs are assigned to the generation levels, which do not satisfy these constraints.

4.1.5. Back-track Operation

The back-track operation is the simplest step of the program. During the calculation procedure for each stage the path from previous hour, which gives minimum total cost, is saved. These paths are then searched from the end to the beginning to find the optimum path and the output levels of the generators. If at the last step all costs are infinite, it means that this program could not find a feasible path over the period with the given generator data, constraints and state list.

4.2. Reduction of State List

Further reductions have to be made in order to reduce computational requirements for high number of generators, since, in this study no heuristic techniques are used in the unit commitment solution to reduce the number of combinations, most of the combinations are calculated during the solution process. Due to the fact that the combinations and the execution time are increasing exponentially with the generator number, where at each stage the the maximum number of combinations is equal to the number of cycling units to the power of two minus 1, the execution time of the program will be unpractical for generator numbers greater than 7. So, for generator numbers greater than 7 (cycling), some heuristic techniques must be used to reduce the number of calculations.

4.3. Results of the Sample System 1

The program is tested with a sample system for verification, of which the solution is known [1, pp. 125]. The input/output data is given in Appendix A. The solution found by the program agrees with the solution in the reference [1, pp. 125]. First the system is tested with neglected minimum up/down times. The second test includes also the minimum up/down times, which changes the commitment schedule of the generators.

In the program with neglected up/down times, complete enumaration is used for cycling units. The found commitment the unit 3 down in the hours 6 and 7, when the load decreases a large amount. Note that, it is less expensive to turn on the less efficient peaking unit, number 4, for hour 3 than to start up the more efficient unit 1 for that period.

In the second case the minimum up/down times are observed. Again complete enumaration is used for cycling units. In this case the real optimum is found, where the cost is higher than the first case because including the up/down times result in an additional cost. The unit 3 can not be shutdown for hours 6 and 7 because of the minimum down time restrictions. The found costs agree exactly with the example in [1, pp. 125]. With this example it is shown that the program can find the true optimum schedule of the unit commitment problem. An example with more constraints added is tested and explained in the next section. Execution times are given in Table 4.1 .

	16MHz 286	Vax 8250
Execution Times (Sec)	61 secs	3 secs

Table 4.1 Execution times for sample system 1

4.4. Results of the Sample System 2

The program is tested with a hypotetical power system in order to check whether the constraints are handled properly. The program is tested for three systems by removing some of the constraints from the system to observe the effect of constraints to the scheduling of generators and cost.

The data of generators and power system for the sample system 2 can be found in Appendix B. Also, The output schedules of the generators for all three cases can be found in Appendix B. The program was able to find a generator scheduling output which satisfies all the constraints with minimum cost.

When ramp rate and reserve constraints are removed from the generator data, the program finds a cheaper commitment schedule. The cost decrease is due to the fact that the schedule does not have to satisfy the reserve requirement and also it does not have to obey ramp rates of generators. These two constraints increases the cost of the of commitment schedule.

In the third case the min up/down time restrictions of generators are also removed and the program finds a solution which is cheaper than the second case. The decrease can be explained by the fact that, the program can now shut down units when the load decreases without any restriction.

	Vax 8250
Execution Times (Sec)	35 MIN

Table 4.2 Execution times for sample system 2

CHAPTER 5

IMPROVEMENT OF THE ALGORITHM USING PARALLEL-COMPUTING

The dynamic programming requires some heuristic techniques in order to reduce the state space for large systems. But this reduction will result in a suboptimal solution as mentioned above. Another improvement in the algorithm without reducing the state space can be done using parallel subprocessors.

The dynamic programming algorithm at each stage or time period, requires calculations of minimal cost and path from previous stage for each state in the current stage. But this calculation is dependent only on the data of the previous stage which is calculated beforehand. Hence, if the calculations for each state at a particular stage or time can be seperated and calculated by a different process using a machine with multi-proccesing capabilities, then the computer resources are used very efficiently, and the time required is decreased considerably without reducing the state space. Most multi-user computer systems, such as VAX 8250, have the capability of creating multiple processes.

The VAX/VMS operating system provides a high performance environment for concurrent multiuser time sharing operations and offers a set of very valuable routines for concurrent programming in implementing with high level languages. The

second program has been implemented using these routines and some other system services, such as interprocess communication services. The VAX/VMS manuals related with these subjects are explained in detail in [26]-[30].

5.1. Application of Parallel-Processing to Dynamic Programming

The parallel implementation of dynamic programming algorithm on a VAX 8250 computer is illustrated in Fig. 5.1. Two additional routines are used to create the parent process and the subprocesses, in addition to the routines of the first program. The other subroutine details and data files are the same as in the Appendix C and D. The parent process decides on responsible subprocesses and states to be evaluated and it is responsible from the data transfer between the subprocesses.

A parent process executing the main program creates sequentially N number of concurrent subprocesses, where each one of these concurrent subprocesses calculates a fraction of states at time t. The communication between subprocesses are achieved by using a previously installed common block. In the VAX/VMS operating system, we can use an installed common block for interprocess communication or for allowing two or more processes to access the same data simultaneously. In addition, common event flags are used to synchronize the events between the parent process and the subprocesses. This

synchronization is especially required for communicating using the installed common block. A psuedo algorithm is given in Figure 5.2. The program is tested with the example given in Appendix D. The results obtained are same as befored but the execution times are different, which are given in Figure 5.3.

The following two routines are used to create the parent process and the subprocesses :

Unit Commitment Parent : The parent process to create N(t) subprocesses at time t and syncronize the data transfer between the processes.

Unit Commitment SubProc : The subprocess created by the parent process to calculate some of the states at a given time t.

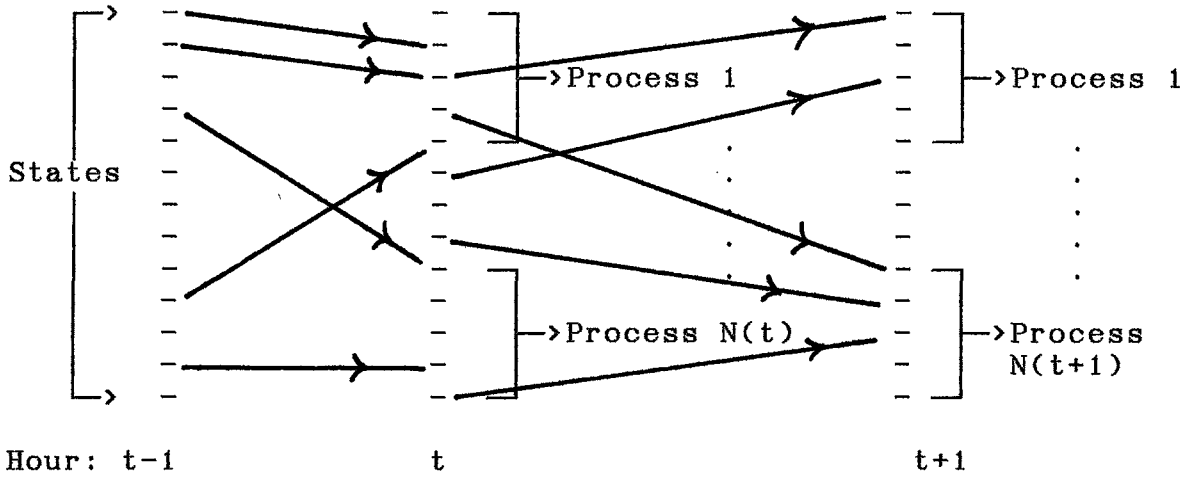


Figure 5.1

```
read input data
eliminate non feasible states from the state list
initialize first hour data
do t = 1 to Max Hour
  determine the number of states at hour t
  determine which subprocess calculates which states
  generate sequentially N subprocesses
  pass to each subprocess (number of states/N ) data
  wait until all processors complete their computation
  read from each process the calculated data
  save the data to use in the next iteration
enddo
Back-Track the optimum path
print the results
```

Pseudo algorithm of parallel-proccessing implementation

Figure 5.2.

Number of Subprocesses	1	2	3	4	8
Execution times (Minute)	35	24	18	15	14

Execution times of the parallel program for sample system 2

Fig 5.3.

5.2. Results of the Parallel-Processing Implementation:

If M is the number of states at time t and N is the number of the processes, each subprocess calculates concurrently M/N states at time t. So theoretically the calculation time decreases N times. But time is needed to create a subprocess and also transfer of data between subprocessors need additional CPU time. The computation time in various multiprocessor organizations remains the same while the communication overhead due to interprocessor data transfer is extremely important. If the number of processors increases more than some limit, the time to create the processes and the time to pass data becomes greater than the time to calculate the states. After that point there is no advantage in creating subprocesses. This process limit is found to be 4 for this particular VAX system. This number is strongly machine dependent and vary from one configuration to another.

CHAPTER 6

CONCLUSION

In this thesis the unit commitment problem is solved with dynamic programming method. Dynamic programming algorithm has the advantage of being able to solve problems of a variety of sizes and complexity (nonlinear,nonconvex,constrained) with high accuracy and without any convergence problems which arises in most of the optimization algorithms. The algorithm can be be easily modified to model characteristics of specific utilites and generators. In order to improve the solution and to simulate a real power system, most of the operational and generator constraints of a power system are added to the algorithm. Additionally, a new solution method to the constrained economic dispatch is proposed and implemented with dynamic programming in order to be able to use nonlinear generator characteristics and to obtain an accurate economic dispatch which effects the sheduling of units and therefore the total cost. The usual approach in most of the papers is to use a simple and fast economic dispatch which results in fast but suboptimal solutions. In the solution of the economic dispatch, a multi-pass search is used to decrease calculation time. With a detailed presearch, some obvious unfeasible states which do not satisfy the demand requirement and the reserve and ready reserve requirements are eliminated from the state list to speed-up the algorithm to a great extend.

The algorithm is tested for two systems for which it can find the true optimum schedules for both cases. The minimized cost includes the production cost, time dependent start-up cost and a constant shutdown cost. The determined commitment schedule satisfies various specified spinning reserve requirements and operating constraints such as ramp rates and minimum up/down times of generators.

Finally, the dynamic programming algorithm on a multi-processor computer system is simulated with a concurrent multi-user system (VAX/VMS) and the communications and timing problems occured and improvements which can be obtained are discussed.

6.1 Discussion

The advantages of dynamic programming is that it is relatively easy to add constraints that affect operations at an hour, such as area constraints. Also the convergence problems with many constraints are minimal when compared to other methods. However, adding the constraints in an optimal manner increases the state list exponentially which requires high CPU times and computer memory which is not practical for many systems. Therefore most of the constraints are handled in economic dispatch problem and not added to the state list, which decreases the calculation time but also results in a suboptimal solution. The constraints which are not included in the state space are minimum up/down times and ramp rates

of generators. Most of the constraints which can be found in practical power systems such as unit high and low limits, unit rate limits, unit minimum up/down times, unit start-up ramps, plant-crew constraints, reserve and readyreserve requirements, area reserve and area ready reserve requirements and unit types, are added to the economic dispatch algorithm and adding constraints will speed-up the process by limiting the search range of dynamic programming. These constraints are handled in economic dispatch problem by eliminating generator outputs which do not satisfy the constraints during the search with dynamic programming. One disadvantage of solving the economic dispatch with dynamic programming is that the time to solve dispatch increases with the number of generators of type cycling and must-run. Therefore a carefull study has to be made to determine which must-run generators can be classified as fixed-output generators. This reduction improves the performance of the dispatch algorithm because fixed-output generators are not included in the search space.

One constraint which is not added is the time dependent parameters of generators. For example the output limits may change in time because of maintenance and failure. Adding this constraint requires only changing variable declarations and does not bring any additional CPU time. Fuel constraints, such as availability, storage problems, flow constraints and multiple-fuel units, severely complicates the unit commitment problem. Therefore, the fuel allocation problem is seperated

from the unit commitment problem which can be solved be a seperate fuel-dispatch algorithm. Another advantage of dynamic programming algorithm is that, it can be interrupted at any time and the algorithm can give the optimum schedule until the time of interrupt.

The solution of unit commitment with dynamic programming requires the calculation of states, which still increase exponentially with the number of generators. So application of dynamic programming for higher than 7 cycling generators for a system with 14 generators is not practical because of high requirements of CPU resources. In the examples given in the appendix, the optimum schedule that is found is the actual optimum because all states are included in the program, however high dimensional systems require application of heuristic methods to reduce the state numbers to reasonable amounts in order to reduce CPU and memory requirements of dynamic programming. Many methods are proposed in several papers which reduces the state space but result in a suboptimal solutions.

The simulation of the parallel-processors results in a decrease in computation time, but the decrease is not reached up to the level of theoretical results because of the concurrent operation of multi-processors created. But in the simulation, the communication problems are solved and the multi-processor system is simulated with satisfactory results. The actual timing improvement will be obtained by

using real multi-processor systems, which are available in the world.

Future advances in unit commitment will come in two areas. First, it is important to find methods that can obtain schedules that are very close to optimum. The availability of cheaper and faster computers with parallel processors should make this possible. The other challenge in unit commitment is to include more aspects of the generating system in the formulation. This includes finding better ways to handle ramp rate constraints, multi-area constraints and fuel constraints as well as special unit types such as dual-boiler units and combined cycle units.

REFERENCES:

[1] A. J. Wood and B. F. Wollenberg, "Power Generation Operation and Control," 1984, John Wiley, New York.

[2] A. I. Cohen and M. Yoshimura, "A branch-and-bound algorithm for unit commitment," IEEE Trans. Power App.Syst., vol.PAS - 102, pp. 444-451, Feb. 1983.

[3] H. Habibolahzadeh and J. A. Bubenko, "Application of decomposition techniques to short-term operation planning of hydrothermal power system," IEEE Trans. Power Syst., vol.PWRS -1, pp.41-47, Feb.1986.

[4] G.S. Laurer et al., "Solution of large scale optimal unit commitment problem, "IEEE Trans. Power App.Syst., vol.PAS-101, pp.79-96, Jan.1982

[5] C.K.Pang and M.C.Chen, "Optimal short-term thermal unit commitment," IEEE Trans. Power App. Syst., vol.PAS -99, no. 6, pp. 2060-208, 1980.

[6] F.E. Villaseca and B. Fardanesh, "Fast thermal generation rescheduling," IEEE Trans. Power Syst., vol. PWRS-2, pp.65-71, Feb. 1987.

[7] J.G. Waight, F. Albuyeh, and A. Bose, "Scheduling of generation and reserve margin using dynamic and linear programming," IEEE Trans. Power App. Syst., vol.PAS-100, May 1981.

[8] W. J. Hobbs, "An enchanced dynamic programming approach for unit commitment," IEEE Trans. Power Syst., vol. 3, pp. 1201-1205, Aug. 1988.

[9] F. N. Lee, "Short-term thermal unit commitment," IEEE Trans. Power Syst., vol. 3, pp.421-428, May 1988

[10] W. L. Snyder and H. D. Powell, "Dynamic programming approach to unit commitment," IEEE Trans. Power Syst., vol. PWRS-2, pp.339-350, May 1987.

[11] H. P. V. Meeteren, "Scheduling of generation and allocation of fuel, using dynamic and linear programming," IEEE Trans.Power App. Syst., vol. PAS-103, pp.1562-1568, July 1984.

[12] A.I. Cohen, "An optimization method for load management scheduling," IEEE Trans. Power Syst., vol. 3, pp. 612 -618, May 1988.

[13] P. P. J. Bosh and G. Honderd, "A solution of the unit commitment problem via decomposition and dynamic programming," IEEE Trans. Power App. Syst., vol.PAS-104, pp.1684-1690, July 1985.

[14] J. S. Yang and N. Chen, "Short term hydrothermal coordination using multi-pass dynamic programming," IEEE Trans. Power Syst., vol. 4, pp.1050-1056, Aug. 1989.

[15] A. I. Cohen and V. R. Sherkat, "Optimization-based methods for operations scheduling," IEEE Proc., vol. 75, pp.1574-1591, Dec. 1987.

[16] R. Nieva, A. Inda and I. Guillen, "Lagrangian reduction of search-range for large-scale unit commitment," IEEE Trans. Power Syst., vol.PWRS-2, pp. 465-473, May 1987.

[17] S.V. Eugene, K. Imhof and S. Mukher, "Implementation of a lagrangian based unit commitment problem," IEEE Trans. Power Syst., vol.4, pp.1373-1380, Oct. 1989.

[18] F. Zhuang and F. D. Galiana, "Towards a more rigorous and practical unit commitment by lagrangian relaxation," IEEE Power Syst., vol.3, pp.763-773, May 1988.

[19] T.S. Dillon, K.W. Edwin, H.D. Kochs and R.J. Taud, "Integer programming approach to the problem of optimal unit commitment with probalistic reserve determination," IEEE Trans. Power App. Syst., vol.PAS-97, pp. 2154-166, Nov/Dec 1978.

[20] G.L. Kusic and H.A. Putnam, "Dispatch and unit commitment including commonly owned units," IEEE Trans. Power App. Syst., vol.PAS-104, pp.2408-2412, Sep.1985.

[21] H.P.V. Meeteren, J.L. Farah and W.T. Wynne, "Extensive unit commitment function including fuel allocation and security area constraint," IEEE Trans. Power Syst., vol.PWRS-1, pp. 228-232, Nov. 1986.

[22] M. Piekutowski and I.A. Rose, "A linear programming method for unit commitment incorparating generator congigurations, reserve and flow constraints," IEEE Trans. Power App. Syst., vol. PAS-104, pp. 3510-3516, Dec. 1985.

[23] A. I. Cohen, " A method for solving the fuel constrained unit commitment problem," IEEE Trans Power Syst., vol. PWRS-2, pp.608-614, Aug. 1987.

[24] R. E. Bellman, Dynamic Programming book, Princeton, New Jersey, Princeton University Press, 1957.

[25] D.J. White, Dynamic Programming, San Francisko, Aberdeen University Press, 1969

[26] Programming in VAX Fortran, 1984, VAX/VMS Version 4.0.

[27] Guide to Programming on VAX/VMS (Fortran Edition), 1984, VAX/VMS Version 4.0.

[28] Introduction to VAX/VMS System Routines, 1984, VAX/VMS Version 4.0.

[29] VAX/VMS System Services Reference Manual, 1985, VAX/VMS Version 4.2.

[30] VAX/VMS Run-Time Library Routines Reference Manual, 1985, VAX/VMS Version 4.2.

Appendix A:

Input Output Data of the Sample System 1

Table A-1

The Generator Data

Generators	1	2	3	4
Max Power MW	80	250	300	60
Min Power MW	25	60	75	20
Minimum up time (h)	4	5	5	1
Minimum down time(h)	2	3	4	1
A0 (MBtu/h)	106.5	292.81	342.37	126.0
A1 (MBtu/hMW)	10.440	9.000	8.730	11.900
A2(MBtu/hMW^2)	0	0	0	0
Initial (h) conditions on(+)/off(-)	-5	+8	+8	-6
TS (MBtu)	175.0	200.0	550.0	0.01
AS	1	1	1	1
BS (MBtu)	0	0	0	0
MS ($)	0	0	0	0
SPNMAX (MW)	∞	∞	∞	∞
RESMAX (MW)	∞	∞	∞	∞
Status	Cycling	Cycling	Cycling	Cycling
BaseCost ($)	0	0	0	0
Incremental Cost ($/MW)	0	0	0	0
ShutDownCost($)	0	0	0	0

Fuel Cost : 2 $/MBtu

Table A-2

Load and percentages used in calculation spinning and ready reserve calcuation and the spinning and ready reserve limits for each hour.

h	SR1 %	SR2 %	RR1 %	RR2 %	REQSPN (MW)	REQRED (MW)	LOAD (MW)
1	0	0	0	0	0	0	450
2	0	0	0	0	0	0	530
3	0	0	0	0	0	0	600
4	0	0	0	0	0	0	540
5	0	0	0	0	0	0	400
6	0	0	0	0	0	0	280
7	0	0	0	0	0	0	290
8	0	0	0	0	0	0	500

Table A-3

Time dependent up/down ramp rates of the generators used.

GEN	1		2		3		4	
h	UP RATE (MW)	DOWN RATE (MW)	UP RATE (MW)	DOWN RATE (MW)	UP RATE (MW)	DOWN RATE (MW)	UP RATE (MW)	DOWN RATE (MW)
1	∞	∞	∞	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞	∞	∞	∞
6	∞	∞	∞	∞	∞	∞	∞	∞
7	∞	∞	∞	∞	∞	∞	∞	∞
8	∞	∞	∞	∞	∞	∞	∞	∞

Table A-4

Output of the program with up/down times neglected

h	Gen1	Gen2	Gen3	Gen4	Cost($)	State
1	0	150	300	0	9208	6
2	0	230	300	0	19856	6
3	0	250	300	50	32307	7
4	0	240	300	0	43135	6
5	0	100	300	0	51443	6
6	0	280	0	0	57017	2
7	0	290	0	0	62765	2
8	0	200	300	0	73274	6

Table A-5

Output of the program with up/down times considered

h	Gen1	Gen2	Gen3	Gen4	Cost($)	State
1	0	150	300	0	9208	6
2	0	230	300	0	19856	6
3	0	250	300	50	32307	7
4	0	240	300	0	43135	6
5	0	100	300	0	51443	6
6	0	60	220	0	57635	6
7	0	60	230	0	64001	6
8	0	200	300	0	74110	6

Number of plants = 0 Number of Areas = 0

Table A-6

The commitment schedule when up/down times are neglected.

Status Number	Unit Status	Total Capacity	hour 1	2	3	4	5	6	7	8
15	1111	690	*	*	*	*	*	*	*	*
14	1110	630	*	*	*	*	*	*	*	*
7	0111	610	*	*	*	*	*	*	*	*
6	0110	550	*	*	*	*	*	*	*	*
11	1011	440	*	*	*	*	*	*	*	*
13	1101	390	*	*	*	*	*	*	*	*
10	1010	380	*	*	*	*	*	*	*	*
3	0011	360	*	*	*	*	*	*	*	*
12	1100	330	*	*	*	*	*	*	*	*
5	0101	310	*	*	*	*	*	*	*	*
2	0010	300	*	*	*	*	*	*	*	*
4	0100	250	*	*	*	*	*	*	*	*
9	1001	140	*	*	*	*	*	*	*	*
8	1000	80	*	*	*	*	*	*	*	*
1	0001	60	*	*	*	*	*	*	*	*

Table A-7

The commitment schedule when up/down times are considered.

Status Number	Unit Status	Total Capacity	hour 1	2	3	4	5	6	7	8
15	1111	690	*	*	*	*	*	*	*	*
14	1110	630	*	*	*	*	*	*	*	*
7	0111	610	*	*	*	*	*	*	*	*
6	0110	550	*	*	*	*	*	*	*	*
11	1011	440	*	*	*	*	*	*	*	*
13	1101	390	*	*	*	*	*	*	*	*
10	1010	380	*	*	*	*	*	*	*	*
3	0011	360	*	*	*	*	*	*	*	*
12	1100	330	*	*	*	*	*	*	*	*
5	0101	310	*	*	*	*	*	*	*	*
2	0010	300	*	*	*	*	*	*	*	*
4	0100	250	*	*	*	*	*	*	*	*
9	1001	140	*	*	*	*	*	*	*	*
8	1000	80	*	*	*	*	*	*	*	*
1	0001	60	*	*	*	*	*	*	*	*

APPENDIX B:

The input/output data for the sample system 2.

Table B-1

Generator Data

	1	2	3	4	5	6
MaxPowerMW	550	550	520	520	450	450
MinPowerMW	250	250	250	250	130	130
Fixed Out.	500	450	500	450	400	400
Min up (h)	30	30	30	30	20	20
Min down h	15	15	15	15	10	10
A0	100.0	110.00	105.00	115.0	82.0	92.0
A1	1.1285	1.1265	1.1954	1.18543	1.2136	1.1136
A2	.00135	.00132	.00127	.00125	.00148	.00140
initial(h) conditions on+/off-	10	10	8	8	7	7
TS (MBtu)	0.0	0.0	0.0	0.0	0.0	0.0
AS	10	10	10	10	5	5
BS (MBtu)	141	131	1339	113.5	113.5	103.5
MS ($)	0.0	0.0	0.0	0.0	0.0	0.0
SPNMAX(MW)	110.0	110.0	100.0	100.0	100.0	100.0
RESMAX(MW)	20.0	20.0	30.0	30.0	30.0	20.0
Status	FIXED	FIXED	FIXED	FIXED	FIXED	FIXED
BaseCost $	10.0	10.0	15.0	20.0	21.0	40.0
Incremetal Cost($/MW)	0.1	0.2	0.3	0.15	0.35	0.4
DownCost $	10.0	10.0	10.0	10.0	10.0	10.0

Fuel Cost : 2 $/MBtu

Generator data

	7	8	9	10	11	12
MaxPowerMW	320.0	420.0	150.0	190.0	220.0	300.0
MinPowerMW	120.0	130.0	50.0	30.0	30.0	30.0
Fixed out	0	0	0	0	0	0
Min up (h)	5	4	5	5	5	5
Min down h	2	3	1	1	2	2
A0	49.0	56.00	59.00	32.0	42.0	40.0
A1	1.2643	1.1643	1.7015	1.6966	1.6066	1.8526
A2	.00289	.00139	.00282	.00382	.00302	.00393
Initial(h) conditions on+/off-	6	6	8	-5	-4	-5
TS (MBtu)	0.0	0.0	10.0	10.0	10.0	10.0
AS	5	5.5	10	10	10	10
BS (MBtu)	93.5	81.5	51.0	47.0	54.0	57.0
MS ($)	0.0	0.0	50.0	50.0	60.0	50.0
SPNMAX(MW)	70.0	90.0	90.0	90.0	90.0	70.0
RESMAX(MW	20.0	20.0	20.0	20.0	20.0	20.0
Status	MUSTRUN	MUSTRUN	MUSTRUN	CYCLING	CYCLING	CYCLING
BaseCost$	10.0	20.0	20.0	12.0	17.0	9.0
Incremetal Cost($/MW)	0.1	0.13	0.23	0.212	0.321	0.07
DownCost$	10.0	10.0	10.0	10.0	10.0	10.0

Fuel Cost : 2 $/MBtu

Generator data

	13	14	15	16	17	18
MaxPowerMW	130.0	420.0				
MinPowerMW	400.0	110.0				
Fixed Out	0	0				
Min up (h)	5	5				
Min down h	3	3				
A0	60.0	49.00				
A1	1.4526	1.2526				
A2	.00293	.00399				
Initial(h) conditions on+/off-	-5	-6				
TS (MBtu)	10.0	11.0				
AS	11.0	16.0				
BS (MBtu)	59.0	53.0				
MS ($)	50.0	52.0				
SPNMAX(MW)	90.0	95.0				
RESMAX(MW	25.0	30.0				
Status	CYCLING	CYCLING				
BaseCost$	18.0	19.0				
Incremetal Cost($/MW)	0.12	0.123				
DownCost$	10.0	10.0				

Fuel Cost : 2 $/MBtu

Up/down ramp rates (MW) of the generators
as a function of time

Gen	1		2		3		4		5		6		7	
hour	up	dow	up	dow	up	dow	up	dow	up	dow	up	dow	up	dow
1	50	50	50	50	50	50	50	50	50	50	50	50	70	70
2	90	90	70	80	70	70	70	80	70	80	70	80	90	90
3	150	150	100	100	100	100	100	100	100	100	100	100	190	150
4	190	190	150	170	150	150	150	170	150	170	150	170	250	250
5	250	250	200	200	200	200	200	200	200	200	200	200	300	300
6	300	300	300	300	250	250	250	300	250	300	250	300	300	300
7	300	300	300	300	300	300	300	300	300	300	300	300	300	300
8	300	300	300	300	300	300	300	300	300	300	300	300	300	300

Up/down ramp rates of the generators

Gen	8		9		10		11		12		13		14	
hour	up	dow	up	dow	up	dow	up	dow	up	dow	up	dow	up	dow
1	70	70	70	70	50	50	50	50	50	50	130	130	110	110
2	150	150	90	90	70	70	70	70	70	70	150	150	150	150
3	190	190	150	150	90	90	90	90	90	90	170	170	170	170
4	270	270	150	150	120	120	120	120	120	120	200	200	200	200
5	300	300	150	150	120	120	120	120	150	150	200	200	200	200
6	300	300	150	150	120	120	120	120	150	150	200	200	200	200
7	300	300	150	150	120	120	120	120	150	150	200	200	200	200
8	300	300	150	150	120	120	120	120	150	150	200	200	200	200

Table B-2

Load and percentages used in calculating spinning and ready reserve calcuation and the spinning and ready reserve limits for each hour.

h	SR1 %	SR2 %	RR1 %	RR2 %	REQSPN (MW)	REQRED (MW)	LOAD (MW)	INTCH (MW)
1	2	0	0.9	0	70.2	31.59	3210	300
2	2	0	0.9	0	72.2	32.49	3310	300
3	2	0	0.9	0	73.4	33.03	3370	300
4	2	0	0.9	0	75.0	33.75	3450	300
5	2	0	0.9	0	77.2	34.74	3560	300
6	2	0	0.9	0	81.2	36.54	3660	400
7	2	0	0.9	0	83.2	37.44	3760	400
8	2	0	0.9	0	86.2	38.79	3910	400
9	2	0	0.9	0	89.2	40.14	4060	400
10	2	0	0.9	0	90.0	40.5	4100	400
11	2	0	0.9	0	90.4	40.68	4120	400
12	2	0	0.9	0	86.0	38.7	4000	300
13	2	0	0.9	0	84.6	38.07	3930	300
14	2	0	0.9	0	82.6	37.17	3830	300
15	2	0	0.9	0	78.4	35.28	3620	300
16	2	0	0.9	0	80.2	36.09	3710	300
17	2	0	0.9	0	82.4	37.08	3820	300
18	2	0	0.9	0	84.6	38.07	3930	300
19	2	0	0.9	0	84.0	37.8	3800	400
20	2	0	0.9	0	82.0	36.0	3700	400
21	2	0	0.9	0	81.0	36.45	3650	400
22	2	0	0.9	0	78.2	35.19	3510	400
23	2	0	0.9	0	76.2	34.29	3410	400
24	2	0	0.9	0	75.0	33.75	3350	400

Number of Areas : 1
Generator Numbers in area 1 : 7 , 10 , 12
Reserve Requirement of area 1 : 40.0 MW
ReadyReserve Requirement of area 1 : 60.0 MW

Number of plants : 2
Generator Numbers in plant 1 : 10 , 12
Generator Numbers in plant 2 : 11 , 14

Table B-3

Outputs of generators (MW)

h	1	2	3	4	5	6	7	8	Cost($)	State
1	500	450	500	450	400	400	310	420	2767.967	17
2	500	450	500	450	400	400	320	420	5828.80	17
3	500	450	500	450	400	400	320	420	9193.62	18
4	500	450	500	450	400	400	310	420	12698.74	18
5	500	450	500	450	400	400	320	420	16545.29	18
6	500	450	500	450	400	400	320	420	21221.06	20
7	500	450	500	450	400	400	320	420	26150.12	20
8	500	450	500	450	400	400	320	420	31581.56	20
9	500	450	500	450	400	400	320	420	37718.81	24
10	500	450	500	450	400	400	320	420	43935.83	24
11	500	450	500	450	400	400	320	420	50228.12	24
12	500	450	500	450	400	400	320	420	55635.27	20
13	500	450	500	450	400	400	320	420	60795.63	20
14	500	450	500	450	400	400	320	420	65626.70	20
15	500	450	500	450	400	400	300	420	69782.33	20
16	500	450	500	450	400	400	320	420	74221.45	20
17	500	450	500	450	400	400	320	420	79019.98	20
18	500	450	500	450	400	400	320	420	84180.34	20
19	500	450	500	450	400	400	320	420	89224.87	20
20	500	450	500	450	400	400	320	420	93978.55	20
21	500	450	500	450	400	400	320	420	98485.75	18
22	500	450	500	450	400	400	320	420	102485.6	18
23	500	450	500	450	400	400	310	420	106172.4	18
24	500	450	500	450	400	400	300	420	109664.5	18

Output of the generators for sample system 2

Outputs of generators(MW)

h	9	10	11	12	13	14	15	16	Cost($)	State
1	50	30								
2	90	80								
3	60	10				110				
4	50	50				220				
5	50	60				310				
6	50	50			130	390				
7	50	50			210	410				
8	50	50			350	420				
9	90	80		30	400	420				
10	110	100		30	400	420			SAME	SAME
11	120	110		30	400	420			WITH	WITH
12	50	50			340	420			THE	THE
13	50	50			270	420			TABLE	TABLE
14	50	50			200	390			ABOVE	ABOVE
15	50	30			130	290				
16	50	30			190	300				
17	50	50			190	390				
18	50	50			270	420				
19	50	60			230	420				
20	50	50			180	380				
21	100	90				420				
22	50	50				370				
23	50	50				280				
24	50	30				250				

Output of the generators for sample system 2.

Table B-4

Output of the program.

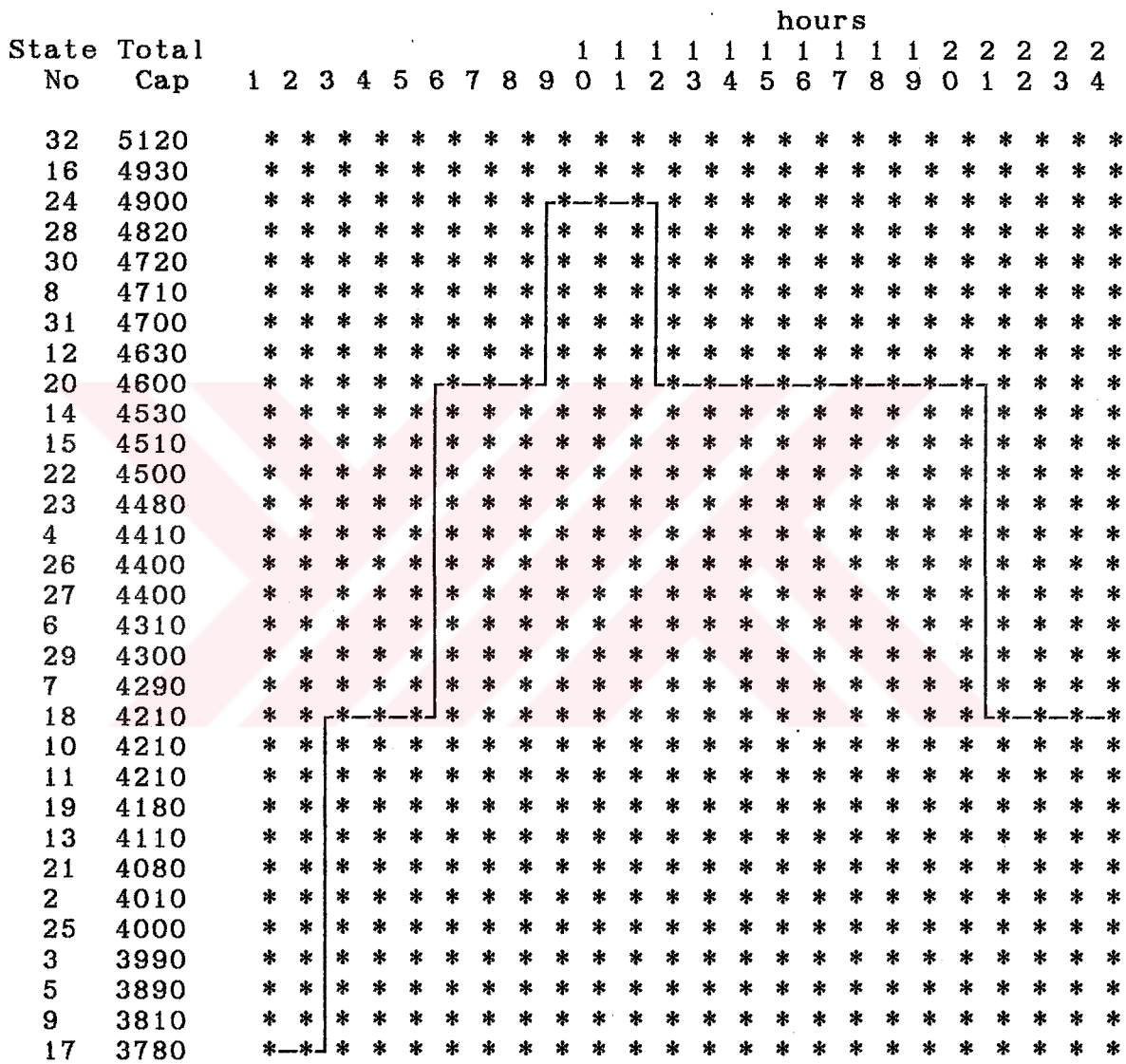


Table B-5

Outputs of generators (MW)

h	1	2	3	4	5	6	7	8	Cost($)	State
1	500	450	500	450	400	400	290	420	2781.785	17
2	500	450	500	450	400	400	320	420	5842.620	17
3	500	450	500	450	400	400	270	420	9170.594	18
4	500	450	500	450	400	400	300	420	12662.67	18
5	500	450	500	450	400	400	320	420	16509.22	18
6	500	450	500	450	400	400	320	420	21175.01	20
7	500	450	500	450	400	400	320	420	26104.07	20
8	500	450	500	450	400	400	320	420	31535.51	20
9	500	450	500	450	400	400	320	420	37680.64	28
10	500	450	500	450	400	400	320	420	43892.86	28
11	500	450	500	450	400	400	320	420	50179.00	28
12	500	450	500	450	400	400	320	420	55586.14	20
13	500	450	500	450	400	400	320	420	60746.50	20
14	500	450	500	450	400	400	320	420	65577.57	20
15	500	450	500	450	400	400	300	420	69733.20	20
16	500	450	500	450	400	400	320	420	74172.31	20
17	500	450	500	450	400	400	320	420	78970.85	20
18	500	450	500	450	400	400	320	420	84131.21	20
19	500	450	500	450	400	400	320	420	89198.73	20
20	500	450	500	450	400	400	320	420	93929.42	20
21	500	450	500	450	400	400	320	420	98436.62	18
22	500	450	500	450	400	400	320	420	102436.5	18
23	500	450	500	450	400	400	310	420	106123.3	18
24	500	450	500	450	400	400	300	420	109615.3	18

Output of the program for sample system 2
without ramp and reserve constraints

Outputs of generators(MW)

h	9	10	11	12	13	14	15	16	Cost($)	State
1	50	50								
2	90	80								
3	50	30				200				
4	50	30				250				
5	50	60				310				
6	50	30			170	370				
7	50	50			210	410				
8	50	50			350	420				
9	50	50	100		400	420				
10	50	60	130		400	420			SAME	SAME
11	60	60	140		400	420			WITH	WITH
12	50	50			340	420			THE	THE
13	50	50			270	420			TABLE	TABLE
14	50	50			200	390			ABOVE	ABOVE
15	50	30			130	290				
16	50	30			190	300				
17	50	50			190	390				
18	50	50			270	420				
19	50	60			230	420				
20	50	50			180	380				
21	100	90				420				
22	50	50				370				
23	50	50				280				
24	50	30				250				

Output of the program for sample system 2
without ramp and reserve constraints

Table B-6

Output of the program.

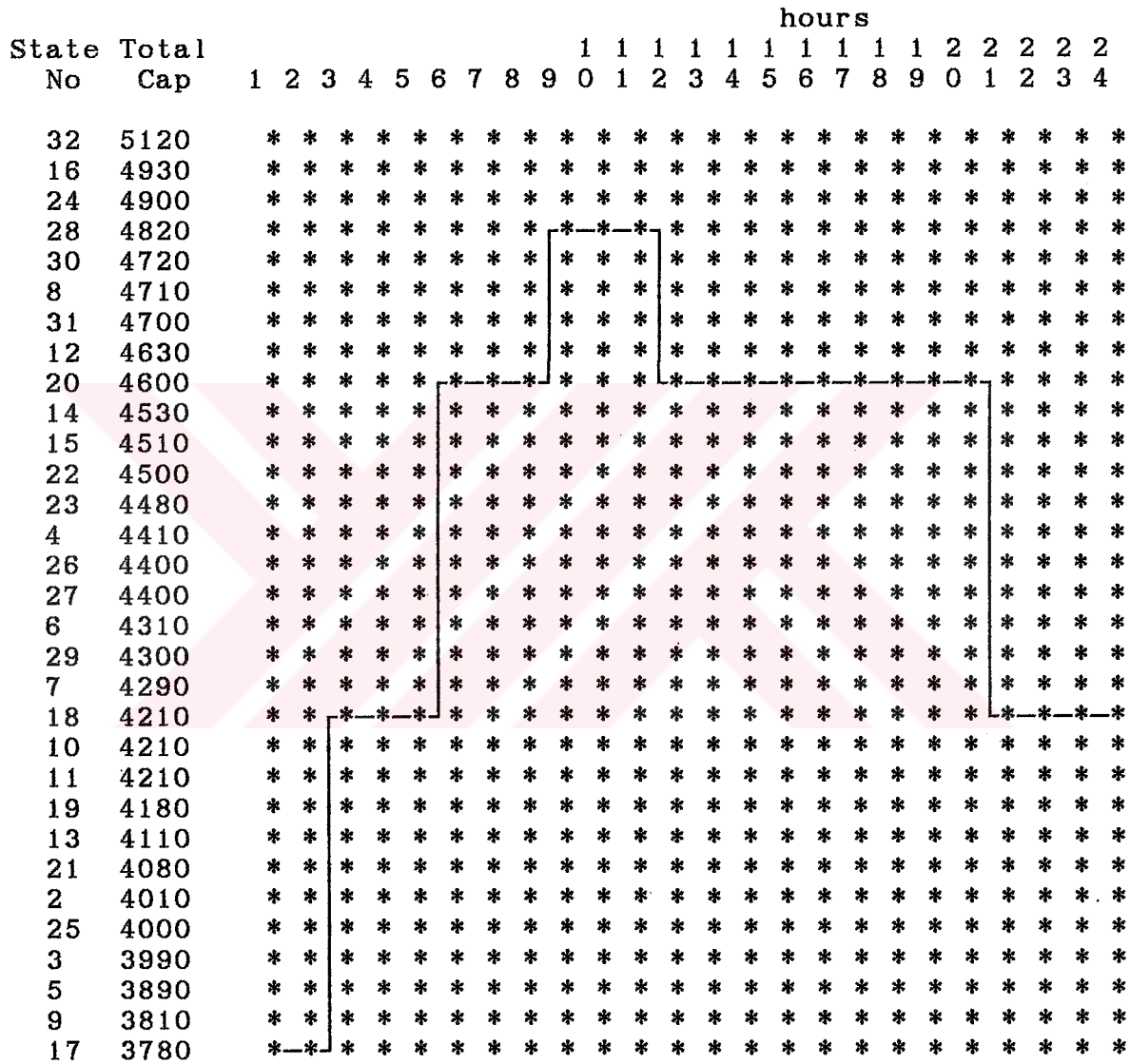


Table B-7

Outputs of generators (MW)

h	1	2	3	4	5	6	7	8	Cost($)	State
1	500	450	500	450	400	400	310	420	2767.967	17
2	500	450	500	450	400	400	320	420	5828.802	17
3	500	450	500	450	400	400	320	420	9193.623	18
4	500	450	500	450	400	400	310	420	1298.747	18
5	500	450	500	450	400	400	320	420	16545.29	18
6	500	450	500	450	400	400	320	420	21221.06	20
7	500	450	500	450	400	400	320	420	26150.12	20
8	500	450	500	450	400	400	320	420	31581.36	20
9	500	450	500	450	400	400	320	420	37718.81	24
10	500	450	500	450	400	400	320	420	43935.83	24
11	500	450	500	450	400	400	320	420	50228.13	24
12	500	450	500	450	400	400	320	420	55635.27	20
13	500	450	500	450	400	400	320	420	60795.63	20
14	500	450	500	450	400	400	320	420	65626.70	20
15	500	450	500	450	400	400	300	420	69668.88	18
16	500	450	500	450	400	400	320	420	74016.84	28
17	500	450	500	450	400	400	320	420	78901.75	20
18	500	450	500	450	400	400	320	420	84062.11	20
19	500	450	500	450	400	400	320	420	89126.63	20
20	500	450	500	450	400	400	320	420	93860.32	20
21	500	450	500	450	400	400	320	420	98367.52	18
22	500	450	500	450	400	400	320	420	102367.4	18
23	500	450	500	450	400	400	310	420	106054.2	18
24	500	450	500	450	400	400	300	420	109546.2	18

Output of the program for sample system 2
without up/down times and reserve constraints

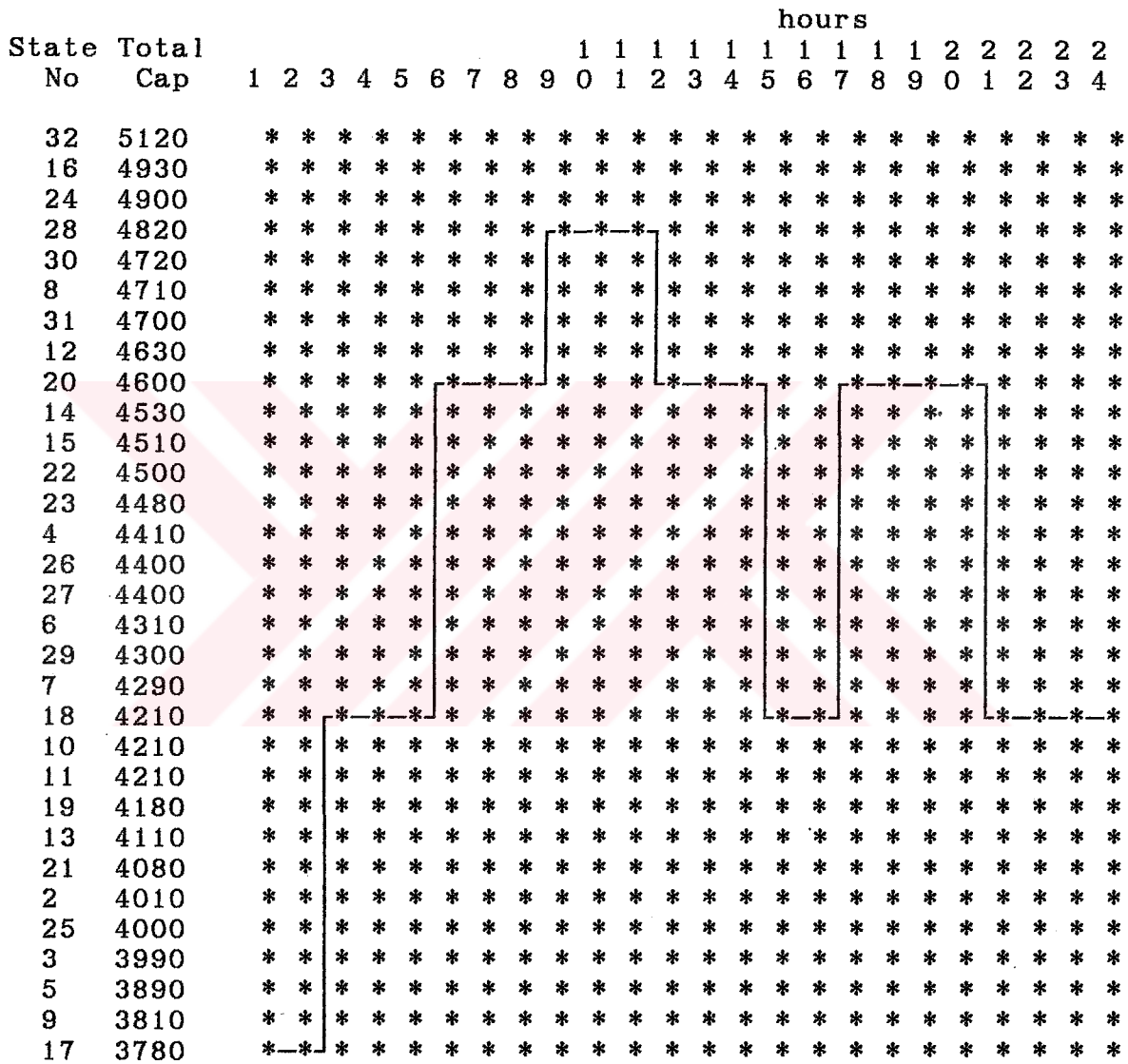
Outputs of generators(MW)

h	9	10	11	12	13	14	15	16	Cost($)	State
1	50	30								
2	90	80								
3	60	60				110				
4	50	50				220				
5	50	60				310				
6	50	50			130	390				
7	50	50			210	410				
8	50	50			350	420				
9	90	80	100		400	420				
10	110	100	130		400	420			SAME	SAME
11	120	110	140	30	400	420			WITH	WITH
12	50	50		30	340	420			THE	THE
13	50	50		30	270	420			TABLE	TABLE
14	50	50			200	390			ABOVE	ABOVE
15	50	50				380				
16	80	70				420				
17	70	60			130	420				
18	50	50			270	420				
19	50	60			230	420				
20	50	50			180	380				
21	100	90				420				
22	50	50				370				
23	50	50				280				
24	50	30				250				

Output of the program for sample system 2
without up/down times and reserve constraints

Table B-8

Output of the program.



Output of the program for sample system 2
without up/down times and reserve constraints

APPENDIX C:

Explanation of the Data Files

-- LOAD.DAT : The LOAD.DAT file contains the demand levels in an hourly basis. In order to increase the time period of the program add the required load levels to LOAD.DAT. The first line is the data of the first hour, the second line is data of the second hour and so on. In order to add comments, leave a blank between the data and the comment.

e.g. LOAD.DAT for N hours.

```
1200.0  1st hour(This is a comment)
3466.0  2nd hour
1231.0  3rd hour
  .       .
  .       .
  .       .
  .       .
2344.5  Nth hour
```

-- INTCHAN.DAT : This data file contains the net interchange levels in hourly basis. Its structure is same with LOAD.DAT. If the net interchange is going out from the power system to a a different power system, add a minus sign to the interchange value.

-- SR12.DAT,RR12.DAT : These two files contain the percentage levels of the spinning (SR12.DAT) and ready reserve (RR12.DAT) requirements The data is given in an hourly basis. The structure of both files is same. The structure of SR12.DAT is given below:

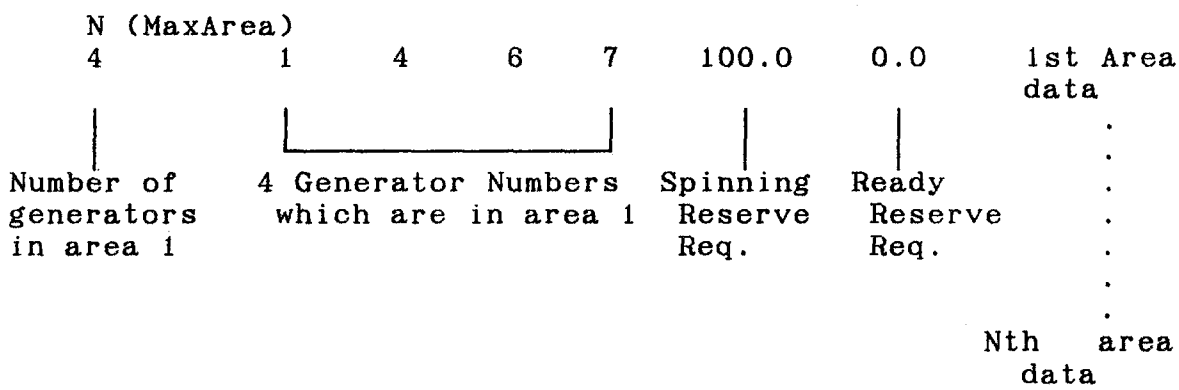
Each line contains two data with a blank between them. The first data is SR1 and the second data is SR2. After leaving a blank after SR2 you can add any comment you like. Also in this file for each hour a value has to be entered.

e.g. SR12.DAT for N hours:

```
10    0 1st hour
10    0     .
 5    0     .
 .    .     .
 .    .     .
 .    .     .
10    0 Nth hour
```

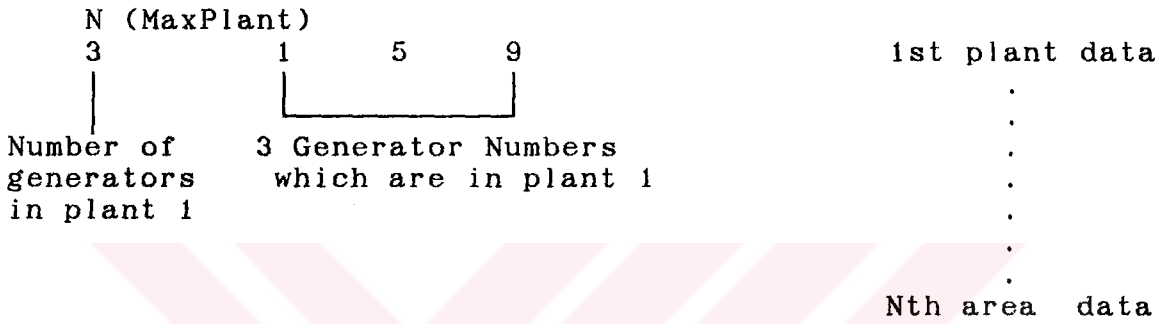
-- AREA.DAT : This file contains the information how the generators are distributed in areas and it also contain the ready and spinning reserve requirements for each area. The first line contains the number or areas. The other lines contains the information for each area.

e.g. AREA.DAT for an area number of N



-- PLANT.DAT : This file contains the information how the generators are distributed in plants. The first line contains the number of plants. The other lines contains the generator lists for each plant.

e.g. PLANT.DAT for a plant number of N



- GEN.DAT : The GEN.DAT contains the operational data and constraints of generators. For each generator a block of data has to be added to the file. Uprate and down rate are given for each hour. The file has the following structure :

GEN.DAT

```
!-------
!-------      ---- 3 lines header for each generator data
!-------
Avalaible    ------ Available/Unavailable ( 1 / 0 )
Type_Of_Gen  ------ Must-Run/Peaker/Cycling/FixedMW (0/1/2/3)
Generation   ------ Generation of FixedMW Generator
Min_Power    ------ Min Power of the generator
Max_Power    ------ Max Power of the generator
A0
A1           ---- The polynomial constants of the heat rate
A2
Min_Up_Time  ------ Minimum up time of the generator
Min_Down_Time ----- Minimum down time of the generator
TS
AS           ---- Parameters for the exponential
BS                start-up cost
MS
Ton/Toff (+/-) ---- The number of hours the unit is on/off
```

Cost_Of_Fuel ——— Cost of the fuel in $
MaxReserveLimit —— Maximum reserve contribution
MaxReadyReserveLimit Maximum ready reserve contribution
BaseMaintenanceCost Base cost of maintenance
IncMaintenanceCost——Incremental cost of maintenance
ShutDownCost ——— Shutdown cost of generator
Uprate ——————— up rate of the generator for each hour
DownRate ——————— down rate of the generator for each hour

This block is repeated for each generator in sequence. The first generator in the list has the number of 1, the second has the number of 2 and so on.

APPENDIX D:

Explanation of the Subroutines

There are several subroutines used in the program. The basic subroutines and their functions are listed below.

Pcost : This subroutine calculates the economic dispatch of a given configuration of generators with dynamic programming, and returns the cost of the economic dispatch. If the required constraints are not satisfied the subroutine returns an infinite value.

Scost : This subroutine returns the startup cost of generators. If the startup of a generator violates a constraint then an infinite cost is returned.

GetAreaReserveLimit : This subroutine calculates the total spinning and ready reserve capacity of a given generator configuration for a given generation level for each area. If the total available spinning and ready reserve limits does not satisfy the required reserve limits, then an infinite cost is returned.

AreaReserveCheck : This subroutine is used in state elimination part of the program, where it eliminates the configurations which does not satisfy the reserve limits.

GetReserveLimit : This subroutine calculates the contribution of a given generator configuration for a given generation

T. C.
Yükseköğretim Kurulu
Dokümantasyon Merkezi

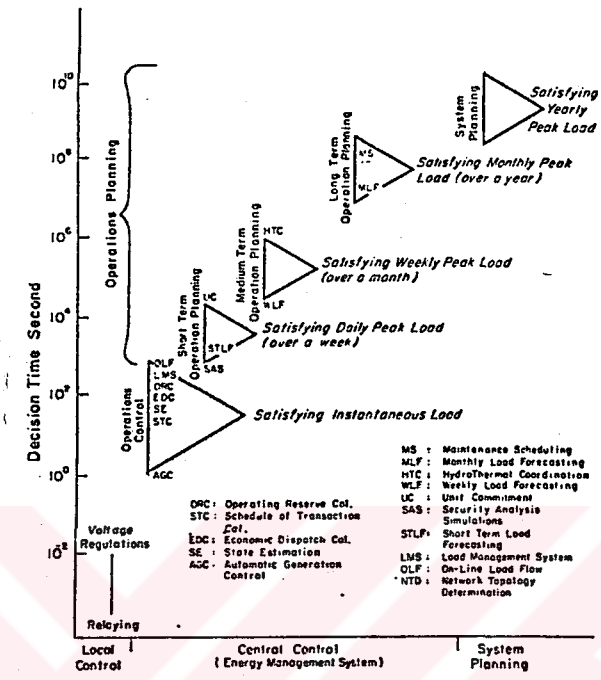
level, and it returns an infinite cost if the reserve capacity of generators does not satisfy the reserve limits.

CheckPlantCrew : This subroutine checks if the plan crew constraints are violated by a transition path, and it returns an infinite cost if the plant constrainst are not satisfied.

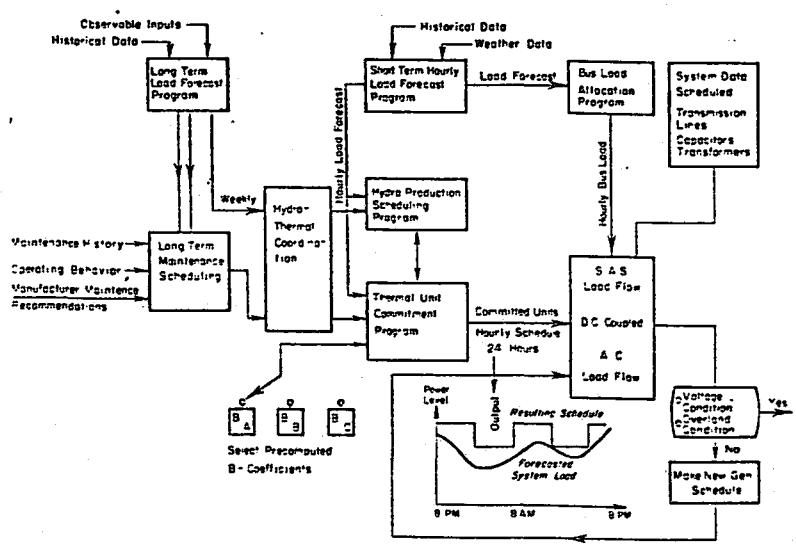
Uc : This is the main program of the simulation and it solves the unit commitment problem with dynamic programming.

APPENDIX E:

Flowchart of operations planning



Energy Management System and Its Functions vs. Decision Time.



Interrelated Tasks for Planned Scheduling Operation.

APPENDIX F:

Simulation of Unit Commitment with Fortran Program

Listing of the Program

```
      Options / Extend_Source

      Program Unit_Commitment
!-----------------------------------------------------------------
      Implicit None
      Include 'UC_Parameter.for'
!-----------------------------------------------------------------
      real MaxP
      real Min_Load
      real PcostVal
      real ScostVal
      real Time
      real Cost
      real Min_Cost
      real PfixedCost
      real MaxReserve
      real MaxReadyReserve
      real MinGeneration
      real MaxGeneration
      real Sr1(1:Max_Hour)
      real Sr2(1:Max_Hour)
      real Rr1(1:Max_Hour)
      real Rr2(1:Max_Hour)
      real Pcost
      real Scost
      real Load(1:Max_Hour)
      real Interchange(1:Max_Hour)
      real P_List(1:Max_State_No,1:Max_Gen_No)

      integer LIB$Get_Lun
      integer LIB$Free_Lun
      integer LIB$Signal
      integer LIB$Show_Timer
      integer LIB$Init_Timer
      integer Status
      integer BinNum(1:20)
      integer Max_Gen
      integer Hour_Cnt
      integer L
      integer State_Cnt
      integer i,j
      integer Min_State
      integer Generator_File
      integer LoadFile
      integer InterchangeFile
      integer SpinReserveFile
      integer ReadyReserveFile
      integer AreaFile
      integer PlantFile
      integer Max_Inc
      integer Gen_No
      integer BinNo
```

```
      integer No
      integer Hour_No
      integer State_no
      integer NumOfCycPeak
      integer OnOff(1:Max_Gen_No)
      integer Binary(1:20)
      integer OnOff_List(1:Max_State_No,1:Max_Gen_No)

      logical AreaCheck
      logical Solution
      logical AreaReserveCheck

      character Dummy*1
!-----------------------------------------------------------------
      Record/P_Min_Max/PMinMax(1:Max_Gen_No,0:Max_Inc_Val)
      Record / Hour_Rec / Hour(0:Max_Hour)
!-----------------------------------------------------------------
      Status = LIB$Get_Lun ( LoadFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
      Open ( UNIT         = LoadFile     ,
     .       FILE         = 'Load.Dat'   ,
     .       FORM         = 'Formatted'  ,
     .       STATUS       = 'Old'        ,
     .       RECORDTYPE   = 'VARIABLE'   ,
     .       ACCESS       = 'Sequential' ,
     .       ORGANIZATION = 'Sequential' )
      do i = 1 , Max_Hour
        Read ( LoadFile , '(F10.0)' ) Load(i)
      enddo
      Close ( LoadFile )
      Status = LIB$Free_Lun ( LoadFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
!-----------------------------------------------------------------
      Status = LIB$Get_Lun ( InterchangeFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
      Open ( UNIT         = InterchangeFile     ,
     .       FILE         = 'Intchan.Dat'   ,
     .       FORM         = 'Formatted'  ,
     .       STATUS       = 'Old'        ,
     .       RECORDTYPE   = 'VARIABLE'   ,
     .       ACCESS       = 'Sequential' ,
     .       ORGANIZATION = 'Sequential' )
      do i = 1 , Max_Hour
        Read ( InterchangeFile , '(F10.0)' ) Interchange(i)
      enddo
      Close ( InterchangeFile )
      Status = LIB$Free_Lun ( InterchangeFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
```

```
      Status = LIB$Get_Lun ( SpinReserveFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
      Open ( UNIT         = SpinReserveFile     ,
     .       FILE         = 'SR12.Dat'    ,
     .       FORM         = 'Formatted'   ,
     .       STATUS       = 'Old'         ,
     .       RECORDTYPE   = 'VARIABLE'    ,
     .       ACCESS       = 'Sequential'  ,
     .       ORGANIZATION = 'Sequential' )
      do i = 1 , Max_Hour
        Read ( SpinReserveFile , '(2F5.0)' )Sr1(i),Sr2(i)
      enddo
      Close ( SpinReserveFile )
      Status = LIB$Free_Lun ( SpinReserveFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
!-----------------------------------------------------------------
      Status = LIB$Get_Lun ( ReadyReserveFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
      Open ( UNIT         = ReadyReserveFile     ,
     .       FILE         = 'RR12.Dat'    ,
     .       FORM         = 'Formatted'   ,
     .       STATUS       = 'Old'         ,
     .       RECORDTYPE   = 'VARIABLE'    ,
     .       ACCESS       = 'Sequential'  ,
     .       ORGANIZATION = 'Sequential' )
      do i = 1 , Max_Hour
        Read ( ReadyReserveFile , '(2F5.0)' )Rr1(i),Rr2(i)
      enddo
      Close ( ReadyReserveFile )
      Status = LIB$Free_Lun ( ReadyReserveFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
!-----------------------------------------------------------------
      Status = LIB$Get_Lun ( AreaFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
      Open ( UNIT         = AreaFile     ,
     .       FILE         = 'Area.Dat'    ,
     .       FORM         = 'Formatted'   ,
     .       STATUS       = 'Old'         ,
     .       RECORDTYPE   = 'VARIABLE'    ,
     .       ACCESS       = 'Sequential'  ,
     .       ORGANIZATION = 'Sequential' )
      Read ( AreaFile , '(I10)' ) MaxArea
      do i = 1 , MaxArea
        Read ( AreaFile , '(I10)' ) Area(i).MaxList
        Read ( AreaFile , '(<Area(i).MaxList>I5)' )
     .    ( Area(i).Gen_List(j) , j = 1 , Area(i).MaxList)
        Read (AreaFile,'(2F10.0)')Area(i).ReserveReq ,
     .                            Area(i).ReadyReservereq
      enddo
      Close ( AreaFile )
      Status = LIB$Free_Lun ( AreaFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
```

```
      Status = LIB$Get_Lun ( PlantFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
      Open ( UNIT         = PlantFile     ,
     .       FILE         = 'Plant.Dat'   ,
     .       FORM         = 'Formatted'   ,
     .       STATUS       = 'Old'         ,
     .       RECORDTYPE   = 'VARIABLE'    ,
     .       ACCESS       = 'Sequential'  ,
     .       ORGANIZATION = 'Sequential' )
      Read ( PlantFile , '(I10)' ) MaxPlant
      do i = 1 , MaxPlant
        Read ( PlantFile , '(I10)' ) Plant(i).MaxList
        Read ( PlantFile , '(<Plant(i).MaxList>I5)' )
     .    ( Plant(i).Gen_List(j),j=1,Plant(i).MaxList)
      enddo
      Close ( PlantFile )
      Status = LIB$Free_Lun ( PlantFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
!-----------------------------------------------------------------
      Status = LIB$Get_Lun ( Generator_File )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
      Open ( UNIT         = Generator_File     ,
     .       FILE         = 'Gen.Dat'    ,
     .       FORM         = 'Formatted'  ,
     .       STATUS       = 'Old'        ,
     .       RECORDTYPE   = 'VARIABLE'   ,
     .       ACCESS       = 'Sequential' ,
     .       ORGANIZATION = 'Sequential' )
      do i = 1 , Max_Gen_No
          Read ( Generator_File , '(A1)' ) Dummy
          Read ( Generator_File , '(A1)' ) Dummy
          Read ( Generator_File , '(A1)' ) Dummy

          Read(Generator_File,'(I5)')Gen(i).Avalaible
          Read(Generator_File,'(I5)')Gen(i).Type_Of_Gen
          Read(Generator_File,'(F14.0)')Gen(i).Generation
          Read(Generator_File,'(F14.0)') Gen(i).Min_Power
          Read(Generator_File,'(F14.0)') Gen(i).Max_Power

          Read( Generator_File , '(F14.0)' ) Gen(i).A0
          Read( Generator_File , '(F14.0)' ) Gen(i).A1
          Read( Generator_File , '(F14.0)' ) Gen(i).A2

          Read(Generator_File,'(I5)')Gen(i).Min_Up_Time
          Read(Generator_File,'(I5)')Gen(i).Min_Down_Time

          Read( Generator_File , '(F14.0)' ) Gen(i).TS
          Read( Generator_File , '(F14.0)' ) Gen(i).AS
          Read( Generator_File , '(F14.0)' ) Gen(i).BS
          Read( Generator_File , '(F14.0)' ) Gen(i).MS

          Read( Generator_File , '(I5)' ) OnOff(i)
```

```
            Read(Generator_File,'(F14.0)')Gen(i).Cost_Of_Fuel
            Read( Generator_File , '(F14.0)' )
     .        Gen(i).MaxReserveLimit
            Read( Generator_File , '(F14.0)' )
     .        Gen(i).MaxreadyReserveLimit
            Read( Generator_File , '(F14.0)' )
     .        Gen(i).BaseMaintenanceCost
            Read( Generator_File , '(F14.0)' )
     .        Gen(i).IncMaintenanceCost
            Read( Generator_File , '(F14.0)' )
     .        Gen(i).ShutdownCost

            Read( Generator_File , '(<MaxRampHour>F6.0)' )
     .        ( Gen(i).Uprate(j),j=1 , MaxRampHour )
            Read( Generator_File , '(<MaxRampHour>F6.0)' )
     .        ( Gen(i).Downrate(j),j=1 , MaxRampHour )

          Hour(0).State(1).Gen_List(i).OnOffTime = OnOff(i)
          if ( OnOff(i).lt.0 ) then
            Hour(0).State(1).Gen_List(i).Status = 0
             Hour(0).State(1).Gen_List(i).Generation = 0.0
          else
            Hour(0).State(1).Gen_List(i).Status = 1
            Hour(0).State(1).Gen_List(i).Generation =
     .        Gen(i).Max_Power
          endif
        enddo
        Close(Generator_File)
        Status = LIB$Free_Lun ( Generator_File )
        if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
!-------------------------------------------------------------
        No = 0
        NumOfCycPeak = 0
        PFixed = 0
        PFixedCost = 0.0
        do i = 1 , Max_Gen_No
          if ( Gen(i).Avalaible.eq.Avalaible ) then
            if ( Gen(i).Type_Of_Gen.eq.FixedMW ) then
              PFixed = PFixed + Gen(i).Generation
              PFixedCost=PFixedCost+Gen(i).Generation    *
     .                              Gen(i).Inc_Heat_rate *
     .                              Gen(i).Cost_Of_Fuel  +
     .                              Gen(i).No_Load_Cost
            endif
            if ( ( Gen(i).Type_Of_Gen.eq.Cycling ) .or.
     .           ( Gen(i).Type_Of_Gen.eq.Peaker  ) ) then
              NumOfCycPeak = NumOfCycPeak + 1
            endif
```

```
            if ( ( Gen(i).Type_Of_Gen.eq.Cycling ) .or.
     .           ( Gen(i).Type_Of_Gen.eq.Peaker  ) .or.
     .           ( Gen(i).Type_Of_Gen.eq.MustRun  ) ) then
              No = No + 1
              GenListFull(No) = i
            endif
          endif
        enddo
        MaxGenListFull = No
!-------------------------------------------------------------------
        Hour(0).Max_State = 1
!         call Integer_To_Binary ( Binary , Max_Gen_No )

        Hour(0).State(1).Cost = 0
        Hour(0).State(1).Path = 0
!-------------------------------------------------------------------
        Max_Load = Load(1) + Interchange(1)
        Min_Load = Load(1) + Interchange(1)
        do i = 1 , Max_Hour
          Load(i) = Load(i) + Interchange(i)
          if ( Max_Load.lt.Load(i) ) Max_Load = Load(i)
          if ( Min_Load.gt.Load(i) ) Min_Load = Load(i)
        enddo
        Max_Inc = JNINT ( Max_Load / Inc_Val )
!-------------------------------------------------------------------
        do Hour_No = 1 , Max_Hour
          State_No = 1
          do i = 0 , JNINT(1.0*(2**NumOfCycPeak)) - 1
            FastP(State_No,Hour_No) = 0
            call Integer_To_Binary (Binary,i,NumOfCycPeak)
            Hour(Hour_No).State(State_No).Cost = 0.0
            Hour(Hour_No).State(State_No).Path = 0.0
            MaxGenListOn(State_No,Hour_No) = 0
            MaxReserve = 0.0
            MaxReadyReserve = 0.0
            MinGeneration = PFixed
            MaxGeneration = PFixed
            BinNo = 1
            MaxP = MaxInt
            do j = 1 , Max_Gen_No
              GenListOn(State_No,Hour_No,j) = 0
              Hour(Hour_No).State(State_No).
     .          Gen_List(j).OnOffTime = 0
              Hour(Hour_No).State(State_No).
     .          Gen_List(j).Generation = 0
              if(((Gen(j).Type_Of_Gen.eq.Cycling ).or.
     .            ( Gen(j).Type_Of_Gen.eq.Peaker)).and.
     .            ( Gen(j).Avalaible.eq.Avalaible))then
                Hour(Hour_No).State(State_No).
     .            Gen_List(j).Status = Binary(BinNo)
```

```
            if ( ( Binary(BinNo) .eq. 0 ).and.
     .           ( Gen(j).Type_Of_Gen.eq.Peaker))then
              FastP(State_No,Hour_No) =
     .          FastP(State_No,Hour_No)+Gen(j).Max_Power
            endif
            if ( Binary(BinNo) .eq. 1 ) then
              if ( MaxP.lt.Gen(j).Max_Power )
     .          MaxP = Gen(j).Max_Power
              MaxGeneration = MaxGeneration +
     .                        Gen(j).Max_Power
              MinGeneration = MinGeneration +
     .                        Gen(j).Min_Power
              MaxReserve = MaxReserve +
     .                     Gen(j).MaxReserveLimit
              MaxReadyReserve =
     .          MaxReadyReserve +
     .          Gen(j).MaxReserveLimit +
     .          Gen(j).MaxReadyReserveLimit
              MaxGenListOn(State_No,Hour_No) =
     .          MaxGenListOn(State_No,Hour_No) + 1
              GenListOn(State_No,Hour_No,
     .          MaxGenListOn(State_No,Hour_No)) = j
            endif
            BinNo = BinNo + 1
          else if((Gen(j).Type_Of_Gen.eq.MustRun).and.
     .            (Gen(j).Avalaible.eq.Avalaible))then
            Hour(Hour_No).State(State_No).
     .        Gen_List(j).OnOffTime = 1
            Hour(Hour_No).State(State_No).
     .        Gen_List(j).Status = 1
            MaxGeneration = MaxGeneration +
     .        Gen(j).Max_Power
            MinGeneration = MinGeneration +
     .        Gen(j).Min_Power
            MaxReserve = MaxReserve +
     .         Gen(j).MaxReserveLimit
            MaxReadyReserve =
     .         MaxReadyReserve + Gen(j).MaxReserveLimit +
     .         Gen(j).MaxReadyReserveLimit
            MaxGenListOn(State_No,Hour_No) + 1 =
     .         MaxGenListOn(State_No,Hour_No) + 1
            GenListOn(State_No,Hour_No,
     .         MaxGenListOn(State_No,Hour_No)) = j
          else if((Gen(j).Type_Of_Gen.eq.FixedMW).and.
     .           (Gen(j).Avalaible.eq.Avalaible ))then
            Hour(Hour_No).State(State_No).
     .        Gen_List(j).OnOffTime = 1
            Hour(Hour_No).State(State_No).
     .        Gen_List(j).Status = 1
            Hour(Hour_No).State(State_No).
     .        Gen_List(j).Generation = Gen(j).Generation
          else
```

```
                Hour(Hour_No).State(State_No).
     .            Gen_List(j).Status = 0
              endif
            enddo
            ReserveReq(State_No,Hour_No) =
     .        Sr1(Hour_No)*Load(Hour_No) +
     .        Sr2(Hour_No)*MaxP
            ReadyReserveReq(State_No,Hour_No) =
     .        Rr1(Hour_No)*Load(Hour_No) +
     .        Rr2(Hour_No)*MaxP
            AreaCheck =
     .        AreareserveCheck(Hour(Hour_No).State(State_No),
     .                         Hour_No , State_No            )
            if ( .NOT.((MaxGeneration.lt.Load(Hour_No)).or.
     .                 (MinGeneration.gt.Load(Hour_No)).or.
     .                 ( MaxReserve .lt.
     .                 ( ReserveReq(State_No,Hour_No) +
     .                 ReadyReserveReq(State_No,Hour_No))).or.
     .                    ( .NOT.AreaCheck ) ) ) then
              State_No = State_No + 1
            endif
          enddo

          Hour(Hour_No).Max_State = State_No - 1

          if ( (State_No-1).eq.0 ) then
            type *,'No Solution....',' Hour :',Hour_No
            stop
          endif
        enddo
!-------------------------------------------------------------
        call LIB$Init_Timer
!-------------------------------------------------------------
        do Hour_Cnt = 1 , Max_Hour
          type *,'Hour : ',Hour_Cnt
          Solution = .False.
          do State_Cnt = 1 , Hour(Hour_Cnt).Max_State
            Min_State = 0
            Min_Cost = Max_Real
            do L = 1 , Hour(Hour_Cnt-1).Max_State
              if ( Hour(Hour_Cnt-1).State(L).Cost.lt.
     .           (Max_Real-100.0) ) then
                PCostVal =
     .            Pcost ( Hour(Hour_Cnt).State(State_Cnt) ,
     .            Hour(Hour_Cnt-1).State(L) ,
     .            State_Cnt       ,
     .            L               ,
     .            P_List          ,
     .            OnOff_List      ,
     .            Hour_Cnt        ,
     .            Load(Hour_Cnt)  )
```

```
                ScostVal =
     .             Scost ( Hour(Hour_Cnt).State(State_Cnt),
     .             Hour(Hour_Cnt-1).State(L)       )

                Cost = Hour(Hour_Cnt-1).State(L).Cost +
     .            PcostVal + SCostVal

                if ( Cost.lt.Min_Cost ) then
                  Min_Cost = Cost
                  Min_State = L
                endif
              endif ! Max_Real
            enddo ! L
!-----------------------------------------------------------------
            if ( Min_State.ne.0 ) then
              Solution = .True.
            endif
!-----------------------------------------------------------------
            Hour(Hour_Cnt).State(State_Cnt).Path = Min_State
            Hour(Hour_Cnt).State(State_Cnt).Cost = Min_Cost +
     .        PFixedCost
!-----------------------------------------------------------------
            if ( Min_State.ne.0 ) then
              do i = 1 , MaxGenlistFull
                Gen_No = GenListFull(i)
                Hour(Hour_Cnt).State(State_Cnt).
     .            Gen_List(Gen_No).Generation =
     .            P_List(Min_State,Gen_No)
                Hour(Hour_Cnt).State(State_Cnt).
     .            Gen_List(Gen_No).OnOffTime =
     .            OnOff_List(Min_State,Gen_No)
              enddo
            endif
!-----------------------------------------------------------------
          enddo ! state_cnt
          if ( .NOT.Solution ) then
            type *,'Hour : ',Hour_Cnt,'---> NO SOLUTION <---'
            call Exit
          endif
!-----------------------------------------------------------------
        enddo ! Hour_Cnt
!-----------------------------------------------------------------
        call LIB$Show_Timer
```

```
!-------------------------------------------------------------
! trace optimum path
!-------------------------------------------------------------
        Min_State = 0
        Min_Cost = Max_Real
        do State_Cnt = 1 , Hour(Max_Hour).Max_State
          if ( Hour(Max_Hour).State(State_Cnt).Cost.lt.
     .       Min_Cost ) then
            Min_Cost = Hour(Max_Hour).State(State_Cnt).Cost
            Min_State = State_Cnt
          endif
        enddo
!-------------------------------------------------------------
        if ( Min_State.eq.0 ) then
          type *,'No Solution'
          stop
        endif
!-------------------------------------------------------------
        type * , ('Hour : '  , Max_Hour , ' '  ,
     .            'State : ' , Min_State , ' ' ,
     .            'Cost : '  , Min_Cost        )


        do i = 1 , Max_Gen_No
          if ( Hour(Max_Hour).State(Min_State).
     .       Gen_List(i).Status.eq.On ) then
            type *,(i,'-->',Hour(Max_Hour).State(Min_State).
     .         Gen_List(i).Generation)
          endif
        enddo

        do Hour_Cnt = Max_Hour , 2 , -1
          Min_State = Hour(Hour_Cnt).State(Min_State).Path
          Min_Cost = Hour(Hour_Cnt-1).State(Min_State).Cost
            type * , ('Hour : ' , Hour_Cnt-1 , ' ' ,
     .                'State : ' , Min_State , ' ' ,
     .                'Cost : ' , Min_Cost         )
          do i = 1 , Max_Gen_No
            if ( Hour(Hour_Cnt-1).State(Min_State).
     .         Gen_List(i).Status.eq.On ) then
              type *,(i,'-->',Hour(Hour_Cnt-1).State(Min_State).
     .           Gen_List(i).Generation)
            endif
          enddo
        enddo
!-------------------------------------------------------------
        end
```

```
      Options / Extend_Source

      logical Function AreaReserveCheck ( State     ,
     .                                    Hour_No   ,
     .                                    State_No  )
!-----------------------------------------------------------------
      Implicit None
      Include 'UC_Parameter.for'
!-----------------------------------------------------------------
      integer i,j
      integer Gen_No
      integer State_No
      integer Hour_No
      Record / State_Rec / State
!-----------------------------------------------------------------
      AreaReserveCheck = .True.
      do i = 1 , MaxArea
        Area(i).ReserveLimit = 0
        Area(i).ReadyReserveLimit = 0
        do j = 1 , Area(i).MaxList
          Gen_No = Area(i).Gen_List(j)
          if ((Gen(Gen_No).Avalaible.eq.Avalaible).and.
     .          ((Gen(Gen_No).Type_Of_Gen.eq.Peaker).or.
     .            (Gen(Gen_No).Type_Of_Gen.eq.MustRun).or.
     .            (Gen(Gen_No).Type_Of_Gen.eq.Cycling)).and.
     .          ( State.Gen_List(Gen_No).Status.eq.On))then
            Area(i).ReserveLimit = Area(i).ReserveLimit +
     .                             Gen(Gen_No).MaxReserveLimit
            Area(i).ReadyReserveLimit =
     .                      Area(i).ReadyReserveLimit +
     .                      Gen(Gen_No).MaxReadyReserveLimit +
     .                      Gen(Gen_No).MaxReserveLimit
          Endif
        Enddo
        if((Area(i).ReserveLimit.lt.Area(i).ReserveReq).or.
     .    (Area(i).ReadyReserveLimit.lt.
     .    (Area(i).ReadyReserveReq +
     .     Area(i).ReserveReq -
     .     FastP(State_No,Hour_No) ))then
          AreaReserveCheck = .False.
          return
        Endif
      Enddo
      return
      end
```

```
      Options / Extend_Source
      logical Function AreaReserveCheck ( State     ,
     .                                    Hour_No   ,
     .                                    State_No  )
!-----------------------------------------------------------
      Implicit None
      Include 'UC_Parameter.for'
!-----------------------------------------------------------
      integer i,j
      integer Gen_No
      integer State_No
      integer Hour_No
      Record / State_Rec / State
!-----------------------------------------------------------
      AreaReserveCheck = .True.
      do i = 1 , MaxArea
        Area(i).ReserveLimit = 0
        Area(i).ReadyReserveLimit = 0
        do j = 1 , Area(i).MaxList
          Gen_No = Area(i).Gen_List(j)
          if ((Gen(Gen_No).Avalaible.eq.Avalaible).and.
     .          ((Gen(Gen_No).Type_Of_Gen.eq.Peaker).or.
     .            (Gen(Gen_No).Type_Of_Gen.eq.MustRun).or.
     .            (Gen(Gen_No).Type_Of_Gen.eq.Cycling)).and.
     .          ( State.Gen_List(Gen_No).Status.eq.On))then
            Area(i).ReserveLimit = Area(i).ReserveLimit +
     .                             Gen(Gen_No).MaxReserveLimit
            Area(i).ReadyReserveLimit =
     .                      Area(i).ReadyReserveLimit +
     .                      Gen(Gen_No).MaxReadyReserveLimit +
     .                      Gen(Gen_No).MaxReserveLimit
          Endif
        Enddo
        if((Area(i).ReserveLimit.lt.Area(i).ReserveReq).or.
     .    (Area(i).ReadyReserveLimit.lt.
     .    (Area(i).ReadyReserveReq +
     .     Area(i).ReserveReq -
     .     FastP(State_No,Hour_No) )))then
          AreaReserveCheck = .False.
          return
        Endif
      Enddo
      return
      end
```

```
      Options / Extend_Source

      logical Function GetAreaReserveLimit ( No       ,
     .                                       P        ,
     .                                       Demand   ,
     .                                       State    ,
     .                                       State_No ,
     .                                       Hour_No  ,
     .                                       P_Val    )
!-----------------------------------------------------------------------
      Implicit None
      Include 'UC_Parameter.for'
!-----------------------------------------------------------------------
      real D
      real PGenList(1:Max_Gen_No)
      real P_Val(1:Max_Gen_No,0:Max_Inc_Val)

      integer No
      integer P
      integer Demand
      integer State_No
      integer Hour_No
      integer cnt
      integer Gen_No
      integer i,j
!-----------------------------------------------------------------------
      Record / State_Rec / State
!-----------------------------------------------------------------------
      GetAreaReserveLimit = .True.
      Gen_No = GenListOn(State_No,Hour_No,No)
      PGenList(Gen_No) = P
      D = Demand - P
!-----------------------------------------------------------------------
      do cnt = No-1 , 1 , -1
        Gen_No = GenListOn (State_No,Hour_No,cnt)
        PGenList(Gen_No) = P_Val(Gen_No,JNINT(D))*Inc_val
        D = D - P_Val(Gen_No,JNINT(D))
      enddo
!-----------------------------------------------------------------------
      do i = 1 , MaxArea
        Area(i).ReserveLimit = 0
        Area(i).ReadyReserveLimit = 0
        do j = 1 , Area(i).MaxList
          Gen_No = Area(i).Gen_List(j)
          if ( ( Gen(Gen_No).Avalaible .eq. Avalaible ) .and.
     .           ( ( Gen(Gen_No).Type_Of_Gen.eq.Peaker ) .or.
     .             ( Gen(Gen_No).Type_Of_Gen.eq.MustRun ) .or.
     .             ( Gen(Gen_No).Type_Of_Gen.eq.Cycling ) ) .and.
     .           ( State.Gen_List(Gen_No).Status.eq.On ) ) then
```

```
            if ( Gen(Gen_No).Committed ) then
              Area(i).ReserveLimit = Area(i).ReserveLimit +
     .          MIN ( Gen(Gen_No).MaxReserveLimit ,
     .                Gen(Gen_No).Max_Power-PGenList(Gen_No) )
              Area(i).ReadyReserveLimit =
     .          Area(i).ReadyReserveLimit+
     .          MIN ( Gen(Gen_No).MaxReadyReserveLimit +
     .                Gen(Gen_No).MaxReserveLimit ,
     .                Gen(Gen_No).Max_Power-PGenList(Gen_No) )
            else
              Area(i).ReserveLimit = Area(i).ReserveLimit +
     .          Gen(Gen_No).MaxReserveLimit
              Area(i).ReadyReserveLimit =
     .          Area(i).ReadyReserveLimit +
     .          Gen(Gen_No).MaxReadyReserveLimit +
     .          Gen(Gen_No).MaxReserveLimit
            endif
          Endif
        Enddo
        if ( ( Area(i).ReserveLimit.lt.Area(i).ReserveReq ).or.
     .       ( Area(i).ReadyReserveLimit.lt.
     .       ( Area(i).ReadyReserveReq +
     .        Area(i).ReserveReq -
     .        FastP(State_No,Hour_No) ))then
          GetAreaReserveLimit = .False.
          return
        Endif
      Enddo
!-----------------------------------------------------------------------
      return
      end
```

```
      Options / Extend_Source

       logical Function GetReserveLimit ( No       ,
     .                                    Demand   ,
     .                                    P        ,
     .                                    P_Val    ,
     .                                    Hour_No  ,
     .                                    State_No )
!-----------------------------------------------------------------
      Implicit None
      Include 'UC_parameter.for'
      real D
      real Reserve
      real ReadyReserve
      real Pi
      real P_Val(1:Max_Gen_No,0:Max_Inc_Val)
      integer Demand
      integer cnt
      integer Gen_No
      integer No
      integer P
      integer Hour_No
      integer State_No
!-----------------------------------------------------------------
      GetReserveLimit = .True.
      Gen_No = GenListOn(State_No,Hour_No,No)
      Reserve = MIN ( Gen(Gen_No).Max_Power-P*Inc_Val ,
     .                Gen(Gen_No).MaxReserveLimit     )
      ReadyReserve = MIN (Gen(Gen_No).Max_Power-P*Inc_Val,
     .                Gen(Gen_No).MaxReserveLimit      +
     .                Gen(Gen_No).MaxReadyReserveLimit )
      D = Demand - P
      do cnt = No-1 , 1 , -1
        Gen_No = GenListOn(State_No,Hour_No,cnt)
        Pi = P_Val(Gen_No,JNINT(D) )
        Reserve=Reserve+MIN(Gen(Gen_No).Max_Power-Pi*
     .                      Inc_Val,
     .                      Gen(Gen_No).MaxReserveLimit)
        ReadyReserve = ReadyReserve +
     .                 MIN(Gen(Gen_No).Max_Power-Pi*
     .                 Inc_Val,
     .                 Gen(Gen_No).MaxReserveLimit      +
     .                 Gen(Gen_No).MaxReadyReserveLimit )
        D = D - Pi
      enddo
!-----------------------------------------------------------------
      do cnt = No+1 , MaxGenListOn(State_No,Hour_No)
        Gen_No = GenListOn(State_No,Hour_No,cnt)
        Reserve = Reserve + Gen(Gen_No).MaxReserveLimit
        ReadyReserve = ReadyReserve +
     .                 Gen(Gen_No).MaxReserveLimit +
     .                 Gen(Gen_No).MaxReadyReserveLimit
      enddo
```

```
        if ((ReadyReserve.lt.(ReserveReq(State_No,Hour_No)+
     .        ReadyReserveReq(State_No,Hour_No) -
     .        FastP(State_No,Hour_No))).or.
     .        (Reserve.lt.ReserveReq(State_No,Hour_No)))then
          GetReserveLimit = .False.
        endif
!-------------------------------------------------------------
        return
        end
```

```
      Options / Extend_Source

      Subroutine Integer_To_Binary ( Bin_Num , IntNum ,
     .                               NumOfCycPeak     )

      integer count
      integer cnt
      integer Num_Prev
      integer Int_Num
      integer IntNum
      integer NumOfCycPeak
      integer Bin_Num(1:20)
!-----------------------------------------------------------------
      Int_Num = IntNum
      do count = 1 , NumOfCycPeak
        Bin_Num ( count ) = 0
      enddo
      cnt = NumOfCycPeak
      Num_Prev = Int_Num
      do while ( Num_Prev.ne.0 )
        Int_Num = JINT ( Int_Num / 2.0 )
        Bin_Num ( cnt ) = Num_Prev - Int_Num * 2
        Num_Prev = Int_Num
        cnt = cnt - 1
      enddo
!-----------------------------------------------------------------
      return
      end
```

```
      Options / Extend_Source
      real Function Scost ( State , State_Prev )

!-----------------------------------------------------------------
      Implicit None
      Include 'UC_Parameter.for'
!-----------------------------------------------------------------
      integer Gen_No
      real Cost
      real Start_Cost
      integer No
      integer OnOffTime
!-----------------------------------------------------------------
      Record / State_Rec / State
      Record / State_Rec / State_Prev
!-----------------------------------------------------------------
      Cost = 0.0
      do No = 1 , MaxGenListFull
        Gen_No = GenListFull(No)
        OnOffTime = JIABS(State.Gen_List(Gen_No).OnOffTime)
        Cost = ( State.Gen_List(Gen_No).Status *
     .           ( 1 - State_Prev.Gen_List(Gen_No).Status))*
     .           ( Gen(Gen_No).Ts*Gen(Gen_No).Cost_Of_Fuel +
     .           ( 1 - EXP ( -OnOffTime/Gen(Gen_No).AS )) *
     .          Gen(Gen_No).Bs*Gen(Gen_No).Cost_Of_Fuel +
     .          Gen(Gen_No).MS ) +
     .          ( State_Prev.Gen_List(Gen_No).Status *
     .          ( 1 - State.Gen_List(Gen_No).Status )) *
     .          Gen(Gen_No).ShutDownCost +
     .          Cost
      enddo
      Scost = Cost
!-----------------------------------------------------------------
      return
      end
```

```
      Options / Extend_Source

      real Function Pcost  ( State       ,
     .                       StatePrev    ,
     .                       StateNo      ,
     .                       StatePrevNo  ,
     .                       P_List       ,
     .                       OnOff_List   ,
     .                       Hour_No      ,
     .                       LoadVal      )
!-----------------------------------------------------------------
      Implicit None
      Include 'UC_Parameter.for'
!-----------------------------------------------------------------
      Structure / Cost_Structure /
        real Data(0:Max_Inc_Val)
      End Structure

      integer StateNo
      integer StatePrevNo
      integer Hour_No
      integer OnOff_List(1:Max_State_No,1:Max_Gen_No)
      integer Max_Gen_List
      integer Demand
      integer P
      integer i,j
      integer StepSize
      integer Gen_No
      integer Gen_No_Prev
      integer No
      integer PowerRateUp
      integer PowerRateDown
      integer P_last
      integer Rate_Hour
      integer Max_Load_Inc
      integer ListMax(1:Max_Gen_No)
      integer ListMin(1:Max_Gen_No)
      integer ListMaxD(1:Max_Gen_No)
      integer ListMinD(1:Max_Gen_No)
      integer Start_P(1:Max_Gen_No)
      integer End_P(1:Max_Gen_No)
      integer PP,DemandP

      real LoadVal
      real P_List(1:Max_State_No,1:Max_Gen_No)
      real D
      real Cost_Of_Demand
      real Transient_Cost
      real P_Val(1:Max_Gen_No,0:Max_Inc_Val)
      real Cost_Last
      real No_Load_Cost
      real Load
```

```
        real MinP,MaxP

        logical Quit
        logical ReserveCheck
        logical AreaReserveCheck
        logical ZeroCheck
        logical CheckPlantCrew
        logical GetReserveLimit
        logical GetAreaReserveLimit
!-------------------------------------------------------------
        Record / Cost_Structure / Cost(1:2)
        Record / State_Rec / State
        Record / State_Rec / StatePrev
!-------------------initial values----------------------------
        Load = LoadVal - PFixed
        Max_Gen_List = MaxGenListOn ( StateNo,Hour_No)
        Max_Load_Inc = JNINT ( 1.0*Max_Load/Inc_Val)
        StepSize = 8

        if ( .NOT.CheckPlantCrew (State , StatePrev))then
          PCost = Max_Real
          return
        endif
!-------------------------------------------------------------
! Update On/Off times of generators
!-------------------------------------------------------------
        do No = 1 , MaxGenListFull
          Gen_No = GenListFull(No)
          P_List(StatePrevNo,Gen_No) = 0
!-------------------------------------------------------------
! Check Ton/Toff
!-------------------------------------------------------------
          if((StatePrev.Gen_List(Gen_No).Status.eq.Off).and.
     .       ( State.Gen_List(Gen_No).Status.eq.On ) ) then
            if ( (ABS(StatePrev.Gen_List(Gen_No).OnOffTime)-
     .             Gen(Gen_No).Min_Up_Time ) .lt. 0 ) then
              Pcost = Max_Real
              return
            endif
          endif

          if((StatePrev.Gen_List(Gen_No).Status.eq.On ).and.
     .         ( State.Gen_List(Gen_No).Status.eq.Off))then
            if ( (ABS(StatePrev.Gen_List(Gen_No).OnOffTime)-
     .             Gen(Gen_No).Min_Down_Time ) .lt. 0 ) then
              Pcost = Max_Real
              return
            endif
```

```
!-------------------------------------------------------------
! Ignore if Rate Limits not hold
!-------------------------------------------------------------
            if ( ( Gen(Gen_No).Type_Of_Gen.ne.MustRun .or.
     .             Gen(Gen_No).Type_Of_Gen.ne.FixedMW ) .and.
     .           ( Gen(Gen_No).Avalaible.eq.Avalaible )) then
              Rate_Hour =
     .          MIN (StatePrev.Gen_List(Gen_No).OnOffTime ,
     .          MaxRampHour )
              PowerRateDown =
     .          JNINT((StatePrev.Gen_list(Gen_No).Generation-
     .              Gen(Gen_No).DownRate(Rate_Hour))/Inc_Val)
              if ( PowerRateDown.gt.0 ) then
                Pcost = Max_real
                return
              endif
            endif
          endif
!-------------------------------------------------------------
          if((StatePrev.Gen_List(Gen_No).Status.eq.Off).and.
     .         ( State.Gen_List(Gen_No).Status.eq.On))then
            OnOff_List ( StatePrevNo,Gen_No) = 1
          else if((StatePrev.Gen_List(Gen_No).Status.eq.On )
     .      .and.(State.Gen_List(Gen_No).Status.eq.Off))then
            OnOff_List ( StatePrevNo,Gen_No) = -1
          else if((StatePrev.Gen_List(Gen_No).Status.eq.On)
     .      .and.( State.Gen_List(Gen_No).Status.eq.On))then
            OnOff_List ( StatePrevNo,Gen_No) =
     .        StatePrev.Gen_List(Gen_No).OnOffTime + 1
          else if((StatePrev.Gen_List(Gen_No).Status.eq.Off)
     .      .and.(State.Gen_List(Gen_No).Status.eq.Off))then
            OnOff_List ( StatePrevNo,Gen_No) =
     .        StatePrev.Gen_List(Gen_No).OnOffTime - 1
          endif
        enddo
!-------------------------------------------------------------
! if only 1 generator exists
!-------------------------------------------------------------
        if ( Max_Gen_List.eq.1 ) then
!-------------------------------------------------------------
! thing min/max power rating of generator 1
!-------------------------------------------------------------
          Gen_No = GenListOn(StateNo,Hour_No,1)
          Gen(Gen_No).Committed = .True.

          Rate_Hour = MIN ( OnOff_List(StatePrevNo,Gen_No),
     .                      MaxRampHour                    )
          PowerRateDown =
     .      JNINT ( (StatePrev.Gen_List(Gen_No).Generation -
     .              Gen(Gen_No).DownRate(Rate_Hour))/Inc_Val)
```

```
          PowerRateUp =
     .      JNINT ( (StatePrev.Gen_List(Gen_No).Generation +
     .              Gen(Gen_No).UpRate(Rate_Hour))/Inc_Val )
          Start_P(1) =
     .      MAX ( JNINT(Gen(Gen_No).Min_Power/Inc_Val),
     .            PowerRateDown                        )
          End_P(1) =
     .      MIN ( JNINT(Gen(Gen_No).Max_Power/Inc_Val),
     .            PowerRateUp                          )
    !-----------------------------------------------------------
          P = JNINT ( Load/Inc_Val )
          if ((P.le.End_P(1)).and.(P.ge.Start_P(1)))then
            ReserveCheck = GetReserveLimit ( 1       ,
     .                                       P       ,
     .                                       P       ,
     .                                       P_Val   ,
     .                                       Hour_No ,
     .                                       StateNo )
          else
            ReserveCheck = .False.
          endif

          if ( ReserveCheck ) then
            AreaReserveCheck = GetAreaReserveLimit ( 1   ,
     .                                               P   ,
     .                                               0   ,
     .                                               State   ,
     .                                               StateNo,
     .                                               Hour_No,
     .                                               P_Val   )
          else
            AreaReserveCheck = .False.
          endif

          if ( AreaReserveCheck.and.ReserveCheck ) then
            Pcost = ( Gen(Gen_No).A0              +
     .                Gen(Gen_No).A1*P*Inc_Val    +
     .                Gen(Gen_No).A2*P*P*Inc_Val ) *
     .                Gen(Gen_No).Cost_Of_Fuel +
     .                Gen(Gen_No).IncMaintenanceCost*
     .                P*Inc_Val+
     .                Gen(Gen_No).BaseMaintenanceCost
            P_List(StatePrevNo,Gen_No) = Load
            return
          else
            Pcost = Max_real
            return
          endif
        endif
```

```
!-------------------------------------------------------------
! if more than 1 generator exist
!-------------------------------------------------------------
        do No = 1 , Max_Gen_List
          Gen_No = GenListOn(StateNo,Hour_No,No)
          Rate_hour = MIN ( OnOff_List(StatePrevNo,Gen_No) ,
     .                      MaxRampHour                     )
          PowerRateDown =
     .      JNINT ( (StatePrev.Gen_List(Gen_No).Generation -
     .               Gen(Gen_No).DownRate(Rate_Hour))/Inc_Val)
          PowerRateUp =
     .      JNINT ( (StatePrev.Gen_List(Gen_No).Generation +
     .               Gen(Gen_No).UpRate(Rate_Hour))/Inc_Val )
          Start_P(Gen_No) =
     .      MAX ( JNINT(Gen(Gen_No).Min_Power/Inc_Val),
     .            PowerRateDown                       )
          End_P(Gen_No) =
     .      MIN ( JNINT(Gen(Gen_No).Max_Power/Inc_Val),
     .            PowerRateUp                         )
        enddo
!-------------------------------------------------------------
        Quit = .False.
        do while ( .NOT.Quit )
          Quit = .True.
          do No = 1 , Max_Gen_List
            Gen_No = GenListOn(StateNo,Hour_No,No)
            P_Val(Gen_No,0) = 0
            ListMax(Gen_No)=JINT(1.0*End_P(Gen_No)/StepSize)
            ListMin(Gen_No)=JNINT(1.0*Start_P(Gen_No)/
     .                            StepSize           )
            if ( ListMax(Gen_No).eq.0 ) then
              Quit = .False.
              StepSize = JINT(StepSize/2.0)
            else if ( ListMin(Gen_No).eq.0 ) then
              ListMin(Gen_No) = ListMin(Gen_No) + 1
            endif
!-------------------------------------------------------------
            if ( Quit ) then
              ListMaxD(Gen_No) = ListMax(Gen_No)
              ListMinD(Gen_No) = ListMin(Gen_No)
              if ( No.gt.1 ) then
                Gen_No_Prev=GenListOn(StateNo,Hour_No,No-1)
                ListMaxD(Gen_No) = MIN ( ListMaxD(Gen_No) +
     .                                   ListMaxD(Gen_No_Prev) ,
     .                                   JINT(1.0*Max_Inc_Val/
     .                                   StepSize))
                ListMinD(Gen_No) = MIN ( ListMinD(Gen_No) +
     .                                   ListMinD(Gen_No_Prev) ,
     .                                   JINT(1.0*Max_Inc_Val/
     .                                   StepSize))

              endif
            endif ! quit
```

```
!-------------------------------------------------------------
          enddo ! No
        enddo ! While
!-------------------------------------------------------------
        Cost(1).Data(0) = Max_Real
        Cost(2).Data(0) = Max_Real

        do while ( StepSize .ne. 0 )
          D = JINT(1.0*JNINT(Load/Inc_Val)/StepSize)*StepSize
          do No = 1 , Max_Gen_List
            Gen_No = GenListOn ( StateNo,Hour_No,No)
            P_List(StatePrevNo,Gen_No) = 0
            Gen(Gen_No).Committed = .False.
            P_Val(Gen_No,JNINT(D) ) = 0
          Enddo
          Cost(1).Data(0) = Max_Real
          Cost(2).Data(0) = Max_Real
          Cost(1).Data(JNINT(D)) = Max_Real
          Cost(2).Data(JNINT(D)) = Max_Real
!-------------------------------------------------------------
          Gen_No = GenListOn(StateNo,Hour_No,1)
          Gen(Gen_No).Committed = .True.

          Do PP = ListMin(Gen_No) , ListMax(Gen_No)
            P = PP*StepSize
            Cost(1).Data(P) = Max_Real
            P_Val(Gen_No,P) = 0
            ReserveCheck = GetReserveLimit  ( 1        ,
     .                                        P        ,
     .                                        P        ,
     .                                        P_Val    ,
     .                                        Hour_No  ,
     .                                        StateNo  )
            if ( ReserveCheck ) then
              AreaReserveCheck =
     .          GetAreaReserveLimit  ( 1        ,
     .                                 P        ,
     .                                 0        ,
     .                                 State    ,
     .                                 StateNo  ,
     .                                 Hour_No  ,
     .                                 P_Val    )
            else
              AreaReserveCheck = .False.
            endif
```

```
            if ( ReserveCheck .and. AreaReserveCheck ) then
              Cost(1).Data(P)=(Gen(Gen_No).A0              +
     .                        Gen(Gen_No).A1*P*Inc_Val +
     .                        Gen(Gen_No).A2*P*P*Inc_Val) *
     .                        Gen(Gen_No).Cost_Of_Fuel +
     .                        Gen(Gen_No).IncMaintenanceCost*
     .                        P*Inc_Val +
     .                        Gen(Gen_No).BaseMaintenanceCost
              P_Val(Gen_No,P) = P
            endif
          enddo ! PP
!-------------------------------------------------------------
          do No = 2 , Max_Gen_List
            Gen_No = GenListOn ( StateNo,Hour_No,No)
            Gen_No_Prev = GenListOn ( StateNo,Hour_No,No-1)
            Gen(Gen_No).Committed = .True.
!-------------------------------------------------------------
            do DemandP = ListMinD(Gen_No) , ListMaxD(Gen_No)
              Demand = DemandP*StepSize
              Cost(2).Data(Demand) = Max_Real
              P_Val(Gen_No,Demand) = 0.0
              do PP = MAX ( DemandP-ListMaxD(Gen_No_Prev),
     .                       ListMin(Gen_No) ),
     .                  MIN ( DemandP-ListMinD(Gen_No_Prev),
     .                       ListMax(Gen_No) )
                P = PP*StepSize
                if(Cost(1).Data(Demand-P).lt.
     .            Max_Real-100.0) then
                  ReserveCheck =
     .              GetReserveLimit  ( No      ,
     .                                 Demand  ,
     .                                 P       ,
     .                                 P_Val   ,
     .                                 Hour_No ,
     .                                 StateNo )
                else
                  ReserveCheck = .False.
                endif

                  if ( ReserveCheck ) then
                  AreaReserveCheck =
     .              GetAreaReserveLimit  ( No      ,
     .                                     P       ,
     .                                     Demand  ,
     .                                     State   ,
     .                                     StateNo ,
     .                                     Hour_No ,
     .                                     P_Val   )
                else
                  AreaReserveCheck = .False.
                endif
```

```
                if ( AreaReserveCheck.and.ReserveCheck ) then
                  Transient_Cost =
     .              Cost(1).Data(Demand-P)+
     .              ( Gen(Gen_No).A0                +
     .              Gen(Gen_No).A1*P*Inc_Val    +
     .              Gen(Gen_No).A2*P*P*Inc_Val ) *
     .              Gen(Gen_No).Cost_Of_Fuel +
     .              Gen(Gen_No).IncMaintenanceCost*P*Inc_Val+
     .              Gen(Gen_No).BaseMaintenanceCost
                  if(Transient_Cost.lt.
     .              Cost(2).Data(Demand)) then
                    Cost(2).Data(Demand) = Transient_Cost
                    P_Val(Gen_No,Demand) = P
                  endif
                endif
              enddo ! PP
            enddo ! DemandP
            Cost(1) = Cost(2)
          enddo ! No
!-------------------------------------------------------------
          ZeroCheck = .False.
          do i = Max_Gen_List , 1 , -1
            Gen_No = GenListOn(StateNo,Hour_No,i)
            P_List(StatePrevNo,Gen_No) =
     .        P_Val(Gen_No,JNINT(D))*Inc_val
            D = D - P_Val(Gen_No,JNINT(D))
            if ( P_List(StatePrevNo,Gen_No).eq.0.0 )
     .        ZeroCheck = .True.
          enddo

          StepSize = JINT(StepSize/2.0)
          if ( StepSize.eq.0 ) then
            PCost = Cost(2).Data(JNINT(Load/Inc_Val))
            return
          endif
!-------------------------------------------------------------
          if ( .NOT.ZeroCheck ) then
!-------------------------------------------------------------
              do i = 1 , Max_Gen_List
            Gen_No = GenListOn(StateNo,Hour_No,i)
            ListMax(Gen_No) = JINT ( 1.0*
     .        JNINT(1.0*MIN(P_List(StatePrevNo,Gen_No) +
     .        StepSize*Inc_Val,
     .        Inc_Val*End_P(Gen_No))/Inc_Val)/StepSize )
            ListMin(Gen_No) = JINT(1.0*
     .        JNINT(1.0*MAX(P_List(StatePrevNo,Gen_No)-
     .        StepSize*Inc_Val,
     .        Inc_Val*Start_P(Gen_No))/Inc_Val)/StepSize)
            ListMaxD(Gen_No) = ListMax(Gen_No)
            ListMinD(Gen_No) = ListMin(Gen_No)
```

```
              if ( i .gt.1 ) then
                Gen_No_Prev=GenListOn(StateNo,Hour_No,i-1)
                ListMaxD(Gen_No) =
     .            MIN ( ListMaxD(Gen_No) +
     .            ListMaxD(Gen_No_Prev),
     .            JINT(1.0*Max_Inc_Val/StepSize))
                ListMinD(Gen_No) =
     .            MIN ( ListMinD(Gen_No) +
     .            ListMinD(Gen_No_Prev),
     .            JINT(1.0*Max_Inc_Val/StepSize))
              endif
            enddo
!-----------------------------------------------------------------
          else ! zerocheck
!-----------------------------------------------------------------
            Quit = .False.
            do while ( .Not.Quit )
              Quit = .True.
              do No = 1 , Max_Gen_List
                Gen_No = GenListOn(StateNo,Hour_No,No)
                ListMax(Gen_No) =
     .            JINT ( 1.0*End_P(Gen_No)/StepSize )
                ListMin(Gen_No) =
     .            JNINT ( 1.0*Start_P(Gen_No)/StepSize )
                if ( ListMax(Gen_No).eq.0 ) then
                  Quit = .False.
                  StepSize = JINT(StepSize/2.0)
                else if ( ListMin(Gen_No).eq.0 ) then
                  ListMin(gen_No) = ListMin(Gen_No) + 1
                endif
!-----------------------------------------------------------------
                if ( Quit ) then
                  ListMaxD(gen_No) = ListMax(Gen_No)
                  ListMinD(gen_No) = ListMin(Gen_No)
                  if ( No.gt.1 ) then
                    Gen_No_Prev =
     .                GenListOn(StateNo,Hour_No,No-1)
                    ListMaxD(Gen_No) =
     .                MIN ( ListMaxD(Gen_No)+
     .                ListMaxD(Gen_No_Prev),
     .                JINT(1.0*Max_Inc_Val/StepSize)  )
                    ListMinD(Gen_No) =
     .                MIN ( ListMinD(Gen_No)+
     .                ListMinD(Gen_No_Prev),
     .                JINT(1.0*Max_Inc_Val/StepSize)  )
                  endif

                endif ! Quit
              enddo ! No
            Enddo ! while
!-----------------------------------------------------------------
          endif ! zerocheck
```

```
!-----------------------------------------------------------------------
         enddo ! while
        return
        end
```

```
!-------------------------------------------------------------
! UC_PARAMETER.FOR
!-------------------------------------------------------------
        Parameter Max_Area_Limit = 10
        Parameter Max_Plant_Limit = 10
        Parameter Max_Real = 99999999999.9
        Parameter MaxInt = 99999999
        Parameter Max_Gen_No = 14
        Parameter Max_State_No = 32 ! 2^Max_Gen_No-1
        Parameter Max_Inc_Val = 500
        Parameter Max_Hour = 24
        Parameter On = 1
        Parameter Off = 0
        Parameter MustRun = 0
        Parameter Peaker = 1
        Parameter Cycling = 2
        Parameter FixedMW = 3
        Parameter Avalaible = 0
        Parameter UnAvalaible = 1
        Parameter Inc_Val = 10.0
        Parameter MaxRampHour = 7
!-------------------------------------------------------------
        Structure / P_MIN_MAX /
          integer P_Min
          integer P_Max
        End Structure

        Structure / Area_Rec /
          integer Gen_List(1:Max_Gen_No)
          integer MaxList
          real ReserveReq
          real ReadyReserveReq
          real ReserveLimit
          real ReadyReserveLimit
        End Structure

        Structure / Plant_Rec /
          integer Gen_List(1:Max_Gen_No)
          integer MaxList
        End Structure

        Structure / Gen_List_Rec /
          integer Status
          integer OnOffTime
          real Generation
        End Structure

        Structure / State_Rec /
          Record / Gen_List_Rec / Gen_List (1:Max_Gen_No)
          real Cost
          real Path
        End Structure
```

```
      Structure / Hour_Rec /
        Record / State_Rec / State(1:Max_State_No)
        integer Max_State
      End Structure
!-----------------------------------------------------------------
      Structure / Gen_Rec /
        real Generation
        logical Committed
        real Min_Power
        real Max_Power
        real Inc_Heat_Rate
        real No_Load_Cost
        integer Min_Up_Time
        integer Min_Down_Time
        real TS
        real AS
        real BS
        real MS
        real A0
        real A1
         real A2
         real Cost_Of_Fuel
        integer Type_Of_Gen
        integer Avalaible
        real ShutDownCost
        real Uprate(1:Max_Hour)
        real DownRate(1:Max_Hour)
        real MaxReserveLimit
        real BaseMaintenanceCost
        real IncMaintenanceCost
        real MaxReadyReserveLimit
      End Structure

      integer MaxGenListOn ( 1:Max_State_No,1:Max_Hour)
      integer GenListOn(1:Max_State_No,1:Max_Hour,
                        1:Max_Gen_No)
      integer GenListFull(1:Max_Gen_No)
      integer MaxArea
      integer MaxGenListFull
      integer MaxPlant

      real FastP(1:Max_State_No,1:Max_Hour)
      real ReserveReq(1:Max_State_No,1:Max_Hour)
      real ReadyReserveReq(1:Max_State_No,1:Max_Hour)
      real Pfixed
      real Max_Load

      Record / Plant_Rec / Plant(1:Max_Plant_Limit)
      Record / Area_Rec / Area(1:Max_Area_Limit)
      Record / Gen_Rec / Gen(1:Max_Gen_No)
```

```
      Common / UC_Common / Area          ,
     .                     MaxArea       ,
     .                     Plant         ,
     .                     MaxPlant      ,
     .                     GenListFull   ,
     .                     MaxGenListFull,
     .                     GenListOn     ,
     .                     MaxGenListOn  ,
     .                     Gen           ,
     .                     ReserveReq    ,
     .                     ReadyReserveReq,
     .                     FastP         ,
     .                     PFixed        ,
     .                     Max_Load
!---------------------------------------------------------------
```

LISTING OF FORTRAN PROGRAM FOR PARALLEL-PROCESSING SIMULATION

```
        Options / Extend_Source


        Program Unit_Commitment
!-------------------------------------------------------------------
        Implicit Integer ( A - Z )
        Include 'UC_Parameter.for'
        Include '($JPIDEF)'
!-------------------------------------------------------------------
        integer Addr(2)
        integer Ret_Addr(2)
!-------------------------------------------------------------------
        real MaxP
        real Min_Load
        real Cost
        real Min_Cost
        real MaxReserve
        real MaxReadyReserve
        real MinGeneration
        real MaxGeneration
        real Interchange(1:Max_Hour)
        real Sr1(1:Max_Hour)
        real Sr2(1:Max_Hour)
        real Rr1(1:Max_Hour)
        real Rr2(1:Max_Hour)


        character OutStr*10

        logical Calculated(1:MaxProcessNo)
        logical OK
!-------------------------------------------------------------------
        Structure / Quota_Str /
          logical*1 Quota_Item
          integer*4 Quota_Val
        End Structure
        Record / Quota_Str / Quota(1:3)
!-------------------------------------------------------------------
        integer MaxStateSect_Prev
        character MaxStateSect_Prev_Eqv*4
        Equivalence ( MaxStateSect_Prev ,
     .                MaxStateSect_Prev_Eqv )
!-------------------------------------------------------------------
        integer Start_Sect
        integer End_Sect
        character Start_Sect_Eqv*4
        character End_Sect_Eqv*4
        Equivalence ( Start_Sect , Start_Sect_Eqv )
        Equivalence ( End_Sect , End_Sect_Eqv )
!-------------------------------------------------------------------
        integer Process_No
        character Process_No_Eqv*4
```

```
        Equivalence ( Process_No , Process_No_Eqv )
!-------------------------------------------------------------
        integer Hour_Cnt
        character Hour_Cnt_Eqv*4
        Equivalence ( Hour_Cnt , Hour_Cnt_Eqv )
!-------------------------------------------------------------
        integer Start
        integer End
        character Start_Eqv*4
        character End_Eqv*4
        Equivalence ( Start , Start_Eqv )
        Equivalence ( End , End_Eqv )
!-------------------------------------------------------------
        integer Flag_Read
        integer Flag_Write
        character Flag_Read_Eqv*4
        character Flag_Write_Eqv*4
        Equivalence ( Flag_Read , Flag_Read_Eqv )
        Equivalence ( Flag_Write , Flag_Write_Eqv )
!-------------------------------------------------------------
        real Load(1:Max_Hour)
        character*4 Load_Eqv(1:Max_Hour)
        Equivalence ( Load , Load_Eqv )
!-------------------------------------------------------------
        integer LIB$Get_Lun
        integer LIB$Free_Lun
        integer LIB$Signal
        integer LIB$Show_Timer
        integer LIB$Init_Timer
!-------------------------------------------------------------
        Parameter Prio = 8
        Parameter EOF = -1
        Parameter MBX_NAME = 'MAIL_BOX'
        integer*2 MBX_CHAN
        integer*4 MBX_LUN
!-------------------------------------------------------------
        integer Generator_File
        integer LoadFile
        integer InterchangeFile
        integer SpinReserveFile
        integer ReadyReserveFile
        integer AreaFile
        integer PlantFile
!-------------------------------------------------------------
        integer StartSect(1:MaxProcessNo)
        integer EndSect(1:MaxProcessNo)
        integer StartList(1:MaxProcessNo)
        integer EndList(1:MaxProcessNo)
        integer Status
        integer BinNum(1:20)
        integer Max_Gen
        integer L
```

```
        integer State_Cnt
        integer i,j
        integer Min_State
        integer Max_Inc
        integer Gen_No
        integer BinNo
        integer No
        integer Hour_No
        integer State_no
        integer NumOfCycPeak
        integer Binary(1:20)
!-------------------------------------------------------------
        integer OnOff(1:Max_Gen_No)
!-------------------------------------------------------------
        logical Continue
        logical AreaCheck
        logical AreaReserveCheck
!-------------------------------------------------------------
        character Dummy*1
        character*4 OutFile(5) / 'Out1' , 'Out2' , 'Out3' ,
     .                           'Out4' , 'Out5' /
!-------------------------------------------------------------
        Record / P_Min_Max /
     .    PMinMax(1:Max_Gen_No,0:Max_Inc_Val)
        Record / Hour_Rec / Hour(0:Max_Hour)
!-------------------------------------------------------------
        character*(PageFile_Len) Page_Data
        Common / Page_Common / Page_Data
!-------------------------------------------------------------
        Sec_Len=(%LOC(Page_Data(PageFile_Len:PageFile_Len))-
     .              %LOC(Page_Data(1:1) ) + 511 ) / 512
        Addr(1) = %LOC(Page_Data(1:1))
        Addr(2) = %LOC(Page_Data(PageFile_Len:PageFile_Len))
!-------------------------------------------------------------
        Status = LIB$Get_Lun ( LoadFile )
        if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
        Open ( UNIT          = LoadFile     ,
     .         FILE          = 'Load.Dat'   ,
     .         FORM          = 'Formatted'  ,
     .         STATUS        = 'Old'        ,
     .         RECORDTYPE    = 'VARIABLE'   ,
     .         ACCESS        = 'Sequential' ,
     .               ORGANIZATION = 'Sequential' )
        do i = 1 , Max_Hour
          Read ( LoadFile , '(F10.0)' ) Load(i)
        enddo
        Close ( LoadFile )
        Status = LIB$Free_Lun ( LoadFile )
        if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
!-------------------------------------------------------------
```

```
      Status = LIB$Get_Lun ( InterchangeFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
      Open ( UNIT          = InterchangeFile     ,
     .       FILE          = 'Intchan.Dat'   ,
     .       FORM          = 'Formatted'   ,
     .       STATUS        = 'Old'         ,
     .       RECORDTYPE    = 'VARIABLE'    ,
     .       ACCESS        = 'Sequential' ,
     .             ORGANIZATION = 'Sequential' )
      do i = 1 , Max_Hour
        Read ( InterchangeFile , '(F10.0)' ) Interchange(i)
      enddo
      Close ( InterchangeFile )
      Status = LIB$Free_Lun ( InterchangeFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
!-----------------------------------------------------------------
      Status = LIB$Get_Lun ( SpinReserveFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
      Open ( UNIT          = SpinReserveFile     ,
     .       FILE          = 'SR12.Dat'    ,
     .       FORM          = 'Formatted'   ,
     .       STATUS        = 'Old'         ,
     .       RECORDTYPE    = 'VARIABLE'    ,
     .       ACCESS        = 'Sequential' ,
     .             ORGANIZATION = 'Sequential' )
      do i = 1 , Max_Hour
        Read ( SpinReserveFile , '(2F5.0)' ) Sr1(i),Sr2(i)
      enddo
      Close ( SpinReserveFile )
      Status = LIB$Free_Lun ( SpinReserveFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
!-----------------------------------------------------------------
      Status = LIB$Get_Lun ( ReadyReserveFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
      Open ( UNIT          = ReadyReserveFile     ,
     .       FILE          = 'RR12.Dat'    ,
     .       FORM          = 'Formatted'   ,
     .       STATUS        = 'Old'         ,
     .       RECORDTYPE    = 'VARIABLE'    ,
     .       ACCESS        = 'Sequential' ,
     .             ORGANIZATION = 'Sequential' )
      do i = 1 , Max_Hour
        Read ( ReadyReserveFile , '(2F5.0)' )Rr1(i),Rr2(i)
      enddo
      Close ( ReadyReserveFile )
      Status = LIB$Free_Lun ( ReadyReserveFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
!-----------------------------------------------------------------
```

```
      Status = LIB$Get_Lun ( AreaFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
      Open ( UNIT          = AreaFile     ,
     .       FILE          = 'Area.Dat'   ,
     .       FORM          = 'Formatted'  ,
     .       STATUS        = 'Old'        ,
     .       RECORDTYPE    = 'VARIABLE'   ,
     .       ACCESS        = 'Sequential' ,
     .             ORGANIZATION = 'Sequential' )
      Read ( AreaFile , '(I10)' ) MaxArea
      do i = 1 , MaxArea
        Read ( AreaFile , '(I10)' ) Area.A(i).MaxList
        Read ( AreaFile , '(<Area.A(i).MaxList>I5)' )
     .    ( Area.A(i).Gen_List(j),j=1,Area.A(i).MaxList)
        Read ( AreaFile , '(2F10.0)' )
     .    Area.A(i).ReserveReq ,
     .    Area.A(i).ReadyReservereq
      enddo
      Close ( AreaFile )
      Status = LIB$Free_Lun ( AreaFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
!-----------------------------------------------------------------
      Status = LIB$Get_Lun ( PlantFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
      Open ( UNIT          = PlantFile    ,
     .       FILE          = 'Plant.Dat'  ,
     .       FORM          = 'Formatted'  ,
     .       STATUS        = 'Old'        ,
     .       RECORDTYPE    = 'VARIABLE'   ,
     .       ACCESS        = 'Sequential' ,
     .             ORGANIZATION = 'Sequential' )
      Read ( PlantFile , '(I10)' ) MaxPlant
      do i = 1 , MaxPlant
        Read ( PlantFile , '(I10)' ) Plant.P(i).MaxList
        Read ( PlantFile , '(<Plant.P(i).MaxList>I5)' )
     .    ( Plant.P(i).Gen_List(j),j=1,Plant.P(i).MaxList)
      enddo
      Close ( PlantFile )
      Status = LIB$Free_Lun ( PlantFile )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
!-----------------------------------------------------------------
```

```
      Status = LIB$Get_Lun ( Generator_File )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
      Open ( UNIT         =  Generator_File,
     .       FILE         = 'Gen.Dat'     ,
     .       FORM         = 'Formatted'   ,
     .       STATUS       = 'Old'         ,
     .       RECORDTYPE   = 'VARIABLE'    ,
     .       ACCESS       = 'Sequential' ,
     .             ORGANIZATION = 'Sequential' )
      do i = 1 , Max_Gen_No
          Read ( Generator_File , '(A1)' ) Dummy
          Read ( Generator_File , '(A1)' ) Dummy
          Read ( Generator_File , '(A1)' ) Dummy

          Read ( Generator_File , '(I5)' )
     .      Gen.G(i).Avalaible
          Read ( Generator_File , '(I5)' )
     .      Gen.G(i).Type_Of_Gen
          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).Generation
          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).Min_Power
          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).Max_Power

          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).A0
          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).A1
          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).A2

          Read ( Generator_File , '(I5)' )
     .      Gen.G(i).Min_Up_Time
          Read ( Generator_File , '(I5)' )
     .      Gen.G(i).Min_Down_Time

          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).TS
          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).AS
          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).BS
          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).MS

          Read ( Generator_File , '(I5)' ) OnOff(i)
```

```
          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).Cost_Of_Fuel
          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).MaxReserveLimit
          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).MaxreadyReserveLimit
          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).BaseMaintenanceCost
          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).IncMaintenanceCost
          Read ( Generator_File , '(F14.0)' )
     .      Gen.G(i).ShutdownCost

          Read ( Generator_File , '(<MaxRampHour>F6.0)' )
     .      ( Gen.G(i).Uprate(j),j=1 , MaxRampHour )
          Read ( Generator_File , '(<MaxRampHour>F6.0)' )
     .      ( Gen.G(i).Downrate(j),j=1 , MaxRampHour )

        Hour(0).State(1).Gen_List(i).OnOffTime = OnOff(i)
        if ( OnOff(i).lt.0 ) then
          Hour(0).State(1).Gen_List(i).Status = 0
           Hour(0).State(1).Gen_List(i).Generation = 0.0
        else
          Hour(0).State(1).Gen_List(i).Status = 1
          Hour(0).State(1).Gen_List(i).Generation =
     .      Gen.G(i).Max_Power
        endif
      enddo
      Close(Generator_File)
      Status = LIB$Free_Lun ( Generator_File )
      if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
!-----------------------------------------------------------------
      No = 0
      NumOfCycPeak = 0
      PFixed = 0
      PFixedCost = 0.0
      do i = 1 , Max_Gen_No
        if ( Gen.G(i).Avalaible.eq.Avalaible ) then
          if ( Gen.G(i).Type_Of_Gen.eq.FixedMW ) then
            PFixed = PFixed + Gen.G(i).Generation
            PFixedCost=PFixedCost+Gen.G(i).Generation   *
     .                            Gen.G(i).Inc_Heat_rate *
     .                            Gen.G(i).Cost_Of_Fuel  +
     .                            Gen.G(i).No_Load_Cost
          endif
          if ( ( Gen.G(i).Type_Of_Gen.eq.Cycling ) .or.
     .           (Gen.G(i).Type_Of_Gen.eq.Peaker))then
            NumOfCycPeak = NumOfCycPeak + 1
          endif
          if ( ( Gen.G(i).Type_Of_Gen.eq.Cycling ) .or.
     .           (Gen.G(i).Type_Of_Gen.eq.Peaker).or.
     .           (Gen.G(i).Type_Of_Gen.eq.MustRun))then
```

```
              No = No + 1
              GenListFull(No) = i
            endif
          endif
        enddo
        MaxGenListFull = No
!-------------------------------------------------------------
        Hour(0).Max_State = 1

        Hour(0).State(1).Cost = 0
        Hour(0).State(1).Path = 0
!-------------------------------------------------------------
        Max_Load = Load(i) + Interchange(1)
        Min_Load = Load(i) + Interchange(1)
        do i = 1 , Max_Hour
          Load(i) = Load(i) + Interchange(i)
          if ( Max_Load.lt.Load(i) ) Max_Load = Load(i)
          if ( Min_Load.gt.Load(i) ) Min_Load = Load(i)
        enddo
        Max_Inc = JNINT ( Max_Load / Inc_Val )
!-------------------------------------------------------------
        do Hour_No = 1 , Max_Hour
          State_No = 1
          do i = 0 , JNINT(1.0*(2**NumOfCycPeak)) - 1
            FastP(State_No,Hour_No) = 0
            call Integer_To_Binary ( Binary , i , NumOfCycPeak )
            Hour(Hour_No).State(State_No).Cost = 0.0
            Hour(Hour_No).State(State_No).Path = 0.0
            MaxGenListOn(State_No,Hour_No) = 0
            MaxReserve = 0.0
            MaxReadyReserve = 0.0
            MinGeneration = PFixed
            MaxGeneration = PFixed
            BinNo = 1
            MaxP = 0
            do j = 1 , Max_Gen_No
              GenListOn(State_No,Hour_No,j) = 0
              Hour(Hour_No).State(State_No).
     .          Gen_List(j).OnOffTime = 0
              Hour(Hour_No).State(State_No).
     .          Gen_List(j).Generation = 0
              if(((Gen.G(j).Type_Of_Gen.eq.Cycling).or.
     .            (Gen.G(j).Type_Of_Gen.eq.Peaker)).and.
     .            (Gen.G(j).Avalaible.eq.Avalaible))then
                Hour(Hour_No).State(State_No).
     .            Gen_List(j).Status = Binary(BinNo)
                if ( ( Binary(BinNo) .eq. 0 ).and.
     .               ( Gen.G(j).Type_Of_Gen.eq.Peaker))then
                  FastP(State_No,Hour_No) =
     .              FastP(State_No,Hour_No)+
     .              Gen.G(j).Max_Power
                endif
```

```
            if ( Binary(BinNo) .eq. 1 ) then
              if ( MaxP.lt.Gen.G(j).Max_Power )
     .          MaxP = Gen.G(j).Max_Power
              MaxGeneration = MaxGeneration +
     .          Gen.G(j).Max_Power
              MinGeneration = MinGeneration +
     .          Gen.G(j).Min_Power
              MaxReserve = MaxReserve +
     .          Gen.G(j).MaxReserveLimit
              MaxReadyReserve = MaxReadyReserve +
     .           Gen.G(j).MaxReserveLimit +
     .           Gen.G(j).MaxReadyReserveLimit
              MaxGenListOn(State_No,Hour_No) =
     .           MaxGenListOn(State_No,Hour_No) + 1
              GenListOn(State_No,Hour_No,
     .           MaxGenListOn(State_No,Hour_No)) = j
            endif
            BinNo = BinNo + 1
          else if((Gen.G(j).Type_Of_Gen.eq.MustRun).and.
     .            (Gen.G(j).Avalaible.eq.Avalaible))then
            Hour(Hour_No).State(State_No).
     .        Gen_List(j).OnOffTime = 1
            Hour(Hour_No).State(State_No).\
     .        Gen_List(j).Status = 1
            MaxGeneration = MaxGeneration +
     .        Gen.G(j).Max_Power
            MinGeneration = MinGeneration +
     .         Gen.G(j).Min_Power
            MaxReserve = MaxReserve +
     .         Gen.G(j).MaxReserveLimit
            MaxReadyReserve = MaxReadyReserve +
     .         Gen.G(j).MaxReserveLimit +
     .         Gen.G(j).MaxReadyReserveLimit
            MaxGenListOn(State_No,Hour_No) =
     .         MaxGenListOn(State_No,Hour_No) + 1
            GenListOn(State_No,Hour_No,
     .         MaxGenListOn(State_No,Hour_No)) = j
          else if((Gen.G(j).Type_Of_Gen.eq.FixedMW).and.
     .            (Gen.G(j).Avalaible.eq.Avalaible))then
            Hour(Hour_No).State(State_No).
     .        Gen_List(j).OnOffTime = 1
            Hour(Hour_No).State(State_No).
     .        Gen_List(j).Status = 1
            Hour(Hour_No).State(State_No).
     .        Gen_List(j).Generation = Gen.G(j).Generation
          else
            Hour(Hour_No).State(State_No).
     .        Gen_List(j).Status = 0
          endif
        enddo
        ReserveReq(State_No,Hour_No) =
     .    Srl(Hour_No)*Load(Hour_No) +
```

```
     .         Sr2(Hour_No)*MaxP
               ReadyReserveReq(State_No,Hour_No) =
     .         Rr1(Hour_No)*Load(Hour_No) +
     .         Rr2(Hour_No)*MaxP
               AreaCheck = AreareserveCheck
     .                    ( Hour(Hour_No).State(State_No) ,
     .                    Hour_No , State_No             )
               if ( .NOT.((MaxGeneration.lt.Load(Hour_No)).or.
     .                   (MinGeneration.gt.Load(Hour_No)).or.
     .                   ( MaxReserve .lt.
     .                   ( ReserveReq(State_No,Hour_No) +
     .                  ReadyReserveReq(State_No,Hour_No))).or.
     .                     ( .NOT.AreaCheck ) ) ) then
                 State_No = State_No + 1
               endif
             enddo

             Hour(Hour_No).Max_State = State_No - 1

             if ( (State_No-1).eq.0 ) then
               type *,'No Solution....',' Hour :',Hour_No
               stop
             endif
           enddo
!-------------------------------------------------------------
           Status = SYS$ASCEFC ( %VAL(Base_Flag) , 'Cluster',,)
           if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
!-------------------------------------------------------------
           call Create_Page ( 'COMM',Sec_Len,Adrr,Ret_Addr)
!-------------------------------------------------------------
           type *,'Calculation Started'
!-------------------------------------------------------------
           call LIB$Init_Timer
!-------------------------------------------------------------
           do Hour_Cnt = 1 , Max_Hour
!-------------------------------------------------------------
             do i = 1 , MaxProcessNo
               Status=SYS$CLREF(%VAL(Base_Flag+ReadFlagList(i)))
               if (.NOT.Status)call LIB$Signal(%VAL(Status))
               Status = SYS$CLREF ( %VAL(Base_Flag+
     .                              WriteFlagList(i)) )
               if ( .NOT.Status )
     .           call LIB$Signal ( %VAL(Status) )
             enddo
!-------------------------------------------------------------
             type *,'Hour :',Hour_Cnt
!-------------------------------------------------------------
             MaxStateSect_Prev =
     .         JINT(1.0*Hour(Hour_Cnt-1).
     .         Max_State*Total_State_Len/PageFile_Len)+1
             SumVal = 1
             if ( JMOD(Hour(Hour_Cnt).Max_State,
```

```
     .       Max_State_N).eq.0 ) SumVal = 0
             MaxStateSect = JINT (1.0*Hour(Hour_Cnt).
     .       Max_State/Max_State_N)+SumVal

             Max_Process_No = JMIN0(MaxProcessNo,MaxStateSect)
             SectLen = JINT(1.0*MaxStateSect/Max_Process_No)

             StartList(1) = 1
             EndList(1) = SectLen*Max_State_N
             StartSect(1) = 1
             EndSect(1) = JINT(1.0*EndList(1)*Total_State_Len
     .                         /StateRec_Char)+1

             do Process_No = 2 , Max_Process_No - 1
               StartList(Process_No) = StartList(Process_No-1)
     .           + SectLen*Max_State_N
               EndList(Process_No) = EndList(Process_No-1)
     .           + SectLen*Max_State_N
               StartSect(Process_No) =
     .           JINT(1.0*StartList(Process_No)*
     .           Total_State_Len/StateRec_Char)+1
               EndSect(Process_No) =
     .           JINT(1.0*EndList(Process_No)*Total_State_Len/
     .           StateRec_Char)+1
             enddo
             EndList(Max_Process_No) = Hour(Hour_Cnt).Max_State
             if ( Max_Process_No.gt.1 ) then
               StartList(Max_Process_No) =
     .           StartList(Max_Process_No-1) +
     .           SectLen*Max_State_N
               StartSect(Max_Process_No) =
     .           JINT(1.0*StartList(Max_Process_No)*
     .           Total_State_Len/StateRec_Char)+1
               EndSect(Max_Process_No) =
     .           JINT(1.0*EndList(Max_Process_No)*
     .           Total_State_Len/StateRec_Char)+1
             endif
!------------------------------------------------------------
             WriteMask = 0
             do i = 1 , Max_Process_No
               WriteMask = JIBSET ( WriteMask,WriteFlagList(i))
             enddo
!------------------------------------------------------------
             do Process_No = 1 , Max_process_No
!------------------------------------------------------------
!              Status = LIB$GETJPI ( JPI$_BYTLM ,,, OutVal ,
     .                                OutStr , OutLen )
!              type *,'Bytlm:',OutVal
!------------------------------------------------------------
               Start = StartList(Process_No)
               End = EndList(Process_No)
               Start_Sect = StartSect(Process_No)
```

```
            End_Sect = EndSect(Process_No)
            Flag_Read = ReadFlagList(Process_No)
            Flag_Write = WriteFlagList(Process_No)
!-----------------------------------------------------------
            Status = SYS$CLREF ( %VAL(Base_Flag+ReadFlag) )
            if ( .Not.Status ) call LIB$Signal ( %VAL(Status))
            Status = SYS$CLREF ( %VAL(Base_Flag+WriteFlag) )
            if ( .Not.Status ) call LIB$Signal ( %VAL(Status))
!-----------------------------------------------------------
            Status = SYS$CREPRC (,'UCSUB',,
     .        OutFile(Process_No),,,,,%VAL(4),,,)
            if ( .Not.Status ) call LIB$Signal ( %VAL(Status))
!-----------------------------------------------------------
            Page_Data = Hour_Cnt_Eqv //
     .                  MaxArea_Eqv //
     .                  MaxGenListFull_Eqv //
     .                  PFixed_Eqv //
     .                  Max_Load_Eqv //!
     .                  Load_Eqv(Hour_Cnt) //
     .                  Hour(Hour_Cnt-1).Max_State_Eqv //
     .                  MaxStateSect_Prev_Eqv //
     .                  Start_Eqv //
     .                  End_Eqv //
     .                  Start_Sect_Eqv //
     .                  End_Sect_Eqv //
     .                  Flag_Read_Eqv //
     .                  Flag_Write_Eqv //
     .                  Process_No_Eqv //
     .                  PfixedCost_Eqv //
     .                  GenListFull_Eqv

           Status = SYS$SETEF ( %VAL(base_Flag+ReadFlag) )
           if ( .Not.Status )
     .       call LIB$Signal ( %VAL(Status) )
           Status = SYS$WAITFR ( %VAL(base_Flag+WriteFlag))
           if ( .Not.Status) call LIB$Signal(%VAL(Status))
           Status = SYS$CLREF(%VAL(base_Flag+WriteFlag) )
           if ( .Not.Status) call LIB$Signal(%VAL(Status))


           Page_Data(1:Area_len) = Area.Area_Eqv
           Status = SYS$SETEF ( %VAL(base_Flag+ReadFlag) )
           if ( .Not.Status ) call LIB$Signal(%VAL(Status))
           Status = SYS$WAITFR ( %VAL(base_Flag+WriteFlag) )
           if ( .Not.Status ) call LIB$Signal (%VAL(Status))
           Status = SYS$CLREF ( %VAL(base_Flag+WriteFlag) )
           if ( .Not.Status ) call LIB$Signal (%VAL(Status))
```

```
Page_Data(1:Plant_len) = Plant.Plant_Eqv
Status = SYS$SETEF ( %VAL(base_Flag+ReadFlag) )
if ( .Not.Status ) call LIB$Signal(%VAL(Status))
Status = SYS$WAITFR ( %VAL(base_Flag+WriteFlag) )
if ( .Not.Status ) call LIB$Signal(%VAL(Status))
Status = SYS$CLREF ( %VAL(base_Flag+WriteFlag) )
if ( .Not.Status ) call LIB$Signal(%VAL(Status))


do i = 1 , Gen_Arr
  Page_Data(1:Gen_Char) = Gen.Gen_Eqv(i)
  Status = SYS$SETEF ( %VAL(base_Flag+ReadFlag))
  if ( .Not.Status ) call LIB$Signal(%VAL(Status))
  Status = SYS$WAITFR ( %VAL(base_Flag+WriteFlag))
  if ( .Not.Status ) call LIB$Signal(%VAL(Status))
  Status = SYS$CLREF ( %VAL(base_Flag+WriteFlag) )
  if ( .Not.Status ) call LIB$Signal(%VAL(Status))
enddo


do i = 1 , GenListOn_Arr
  Page_Data(1:GenListOn_Char) = GenListOn_Eqv(i)
  Status = SYS$SETEF ( %VAL(base_Flag+ReadFlag))
  if (.Not.Status) call LIB$Signal(%VAL(Status))
  Status = SYS$WAITFR(%VAL(base_Flag+WriteFlag))
  if ( .Not.Status) call LIB$Signal(%VAL(Status))
  Status = SYS$CLREF(%VAL(base_Flag+WriteFlag))
  if (.Not.Status) call LIB$Signal(%VAL(Status))
enddo


do i = 1 , MaxGenListOn_Arr
  Page_Data(1:maxGenListOn_Char) =
    MaxGenListOn_Eqv(i)
  Status = SYS$SETEF ( %VAL(base_Flag+ReadFlag) )
  if ( .Not.Status ) call LIB$Signal(%VAL(Status))
  Status = SYS$WAITFR ( %VAL(base_Flag+WriteFlag))
  if ( .Not.Status ) call LIB$Signal(%VAL(Status))
  Status = SYS$CLREF ( %VAL(base_Flag+WriteFlag) )
  if ( .Not.Status ) call LIB$Signal(%VAL(Status))
enddo


do i = 1 , ReserveReq_Arr
  Page_Data(1:ReserveReq_Char) = ReserveReq_Eqv(i)
  Status = SYS$SETEF ( %VAL(base_Flag+ReadFlag))
  if ( .Not.Status ) call LIB$Signal(%VAL(Status))
  Status = SYS$WAITFR ( %VAL(base_Flag+WriteFlag))
  if ( .Not.Status ) call LIB$Signal(%VAL(Status))
  Status = SYS$CLREF ( %VAL(base_Flag+WriteFlag) )
  if ( .Not.Status ) call LIB$Signal(%VAL(Status))
enddo
```

```
      do i = 1 , ReadyReserveReq_Arr
        Page_Data(1:ReadyReserveReq_Char) =
     .    ReadyReserveReq_Eqv(i)
        Status = SYS$SETEF ( %VAL(base_Flag+ReadFlag))
        if ( .Not.Status ) call LIB$Signal(%VAL(Status))
        Status = SYS$WAITFR ( %VAL(base_Flag+WriteFlag))
        if ( .Not.Status ) call LIB$Signal(%VAL(Status))
        Status = SYS$CLREF ( %VAL(base_Flag+WriteFlag))
        if ( .Not.Status ) call LIB$Signal(%VAL(Status))
      enddo


      do i = 1 , FastP_Arr
        Page_Data(1:FastP_Char) = FastP_Eqv(i)
        Status = SYS$SETEF ( %VAL(base_Flag+ReadFlag) )
        if ( .Not.Status ) call LIB$Signal(%VAL(Status))
        Status = SYS$WAITFR(%VAL(base_Flag+WriteFlag))
        if ( .Not.Status ) call LIB$Signal(%VAL(Status))
        Status = SYS$CLREF(%VAL(base_Flag+WriteFlag))
        if ( .Not.Status ) call LIB$Signal(%VAL(Status))
      enddo


      do i = 1 , MaxStateSect_Prev
        Page_Data(1:StateRec_Char) =
     .    Hour(Hour_Cnt-1).StateRec_Eqv(i)
        Status = SYS$SETEF ( %VAL(base_Flag+ReadFlag))
        if ( .Not.Status ) call LIB$Signal(%VAL(Status))
        Status = SYS$WAITFR(%VAL(base_Flag+WriteFlag))
        if ( .Not.Status ) call LIB$Signal(%VAL(Status))
        Status = SYS$CLREF( %VAL(base_Flag+WriteFlag))
        if ( .Not.Status ) call LIB$Signal(%VAL(Status))
      enddo


      do i = Start_Sect , End_Sect
        Start_Char = (Start-1)*Total_State_Len+1
        End_Char = End*Total_State_Len
        if ( Start_Char.lt.(i-1)*StateRec_Char+1 .or.
     .       Start_Char.ge.i*StateRec_Char ) then
          Start_Char = 1
        else
          Start_Char = JMOD (Start_Char,StateRec_Char )
        endif
        if ( End_Char.lt.(i-1)*StateRec_Char+1 .or.
     .       End_Char.ge.i*StateRec_Char     ) then
          End_Char = StateRec_Char
        else
          End_Char = JMOD ( End_Char , StateRec_Char )
        endif
```

```
            Page_Data(Start_Char:End_Char) =
     .        Hour(Hour_Cnt).StateRec_Eqv(i)
     .        (Start_Char:End_Char)
            Status = SYS$SETEF ( %VAL(base_Flag+ReadFlag) )
            if ( .Not.Status ) call LIB$Signal(%VAL(Status))
            Status = SYS$WAITFR ( %VAL(base_Flag+WriteFlag))
            if ( .Not.Status ) call LIB$Signal(%VAL(Status))
            Status = SYS$CLREF ( %VAL(base_Flag+WriteFlag))
            if ( .Not.Status ) call LIB$Signal(%VAL(Status))
          enddo
!-------------------------------------------------------------
!           type *,'Process Created ..', Process_No
        enddo ! Process_No
!-------------------------------------------------------------
        Continue = .True.
        OK = .False.
!-------------------------------------------------------------
        Process_No = 1
        do while ( .NOT.OK )
!-------------------------------------------------------------
          Status = SYS$WAITFR ( %VAL(Base_Flag+
     .      WriteFlagList(Process_No)) )
          if ( .Not.Status ) call LIB$Signal(%VAL(Status))
          Status = SYS$CLREF ( %VAL(Base_Flag+
     .      WriteFlagList(Process_No)) )
          if ( .Not.Status ) call LIB$Signal(%VAL(Status))
          Start_Sect = StartSect(Process_No)
          End_Sect = EndSect(Process_No)
          Start = StartList(Process_No)
          End = EndList(Process_No)
          Status = SYS$SETEF ( %VAL(Base_Flag+
     .      ReadFlagList(Process_No)) )
          if ( .Not.Status ) call LIB$Signal (%VAL(Status))
!-------------------------------------------------------------
          do i = Start_Sect , End_Sect
            Start_Char = (Start-1)*Total_State_Len+1
            End_Char = End*Total_State_Len
            if ( Start_Char.lt.(i-1)*StateRec_Char+1 .or.
     .           Start_Char.ge.i*StateRec_Char ) then
              Start_Char = 1
            else
              Start_Char = JMOD ( Start_Char ,
     .                            StateRec_Char )
            endif
            if ( End_Char.lt.(i-1)*StateRec_Char+1 .or.
     .           End_Char.ge.i*StateRec_Char ) then
              End_Char = StateRec_Char
            else
              End_Char = JMOD ( End_Char , StateRec_Char )
            endif

            Status = SYS$WAITFR ( %VAL(base_Flag+ReadFlag))
```

```
              if ( .Not.Status ) call LIB$Signal(%VAL(Status))
              Status = SYS$CLREF ( %VAL(base_Flag+ReadFlag))
              if ( .Not.Status ) call LIB$Signal(%VAL(Status))

              Hour(Hour_Cnt).StateRec_Eqv(i)
     .           (Start_Char:End_Char) =
     .           Page_Data(Start_Char:End_Char)

              Status = SYS$SETEF ( %VAL(base_Flag+WriteFlag))
              if ( .Not.Status ) call LIB$Signal(%VAL(Status))
            enddo
!-----------------------------------------------------------------
            Status = SYS$WAITFR ( %VAL(base_Flag+ReadFlag) )
            if ( .Not.Status ) call LIB$Signal(%VAL(Status))
            Status = SYS$CLREF ( %VAL(base_Flag+ReadFlag) )
            if ( .Not.Status ) call LIB$Signal(%VAL(Status))

            NoSolution_Eqv = Page_Data(1:4)

!            type *,'Proc:',Process_No

            if ( NoSolution ) Continue = .False.
            if ( Process_no.ge.Max_Process_No ) OK = .True.
            Process_No = Process_No + 1
          enddo ! OK
!-----------------------------------------------------------------
          Status = SYS$WFLAND ( %VAL(base_Flag) ,
     .      %VAL(WriteMask) )
          if ( .Not.Status ) call LIB$Signal (%VAL(Status))
          if ( .NOT.Continue ) then
            type *,'No Solution. Hour : ',Hour_Cnt
            call Exit
          endif
!-----------------------------------------------------------------
        enddo ! Hour_Cnt
!-----------------------------------------------------------------
        call LIB$Show_Timer
!-----------------------------------------------------------------
        call Close_Page ( Ret_Addr )
        type *,'Solution OK '
        accept *
!-----------------------------------------------------------------
!-----------------------------------------------------------------
        Min_State = 0
        Min_Cost = Max_Real
        do State_Cnt = 1 , Hour(Max_Hour).Max_State
          if ( Hour(Max_Hour).State(State_Cnt).Cost.lt.
     .      Min_Cost ) then
            Min_Cost = Hour(Max_Hour).State(State_Cnt).Cost
            Min_State = State_Cnt
          endif
        enddo
```

```
!-------if ( Min_State.eq.0 ) then-----------------------------
        type *,'No Solution'
        stop
      endif
!-------------------------------------------------------------
      type * , ('Hour : '  , Max_Hour , ' '  ,
     .          'State : ' , Min_State , ' ' ,
     .          'Cost : '  , Min_Cost        )


      do i = 1 , Max_Gen_No
        if ( Hour(Max_Hour).State(Min_State).
     .    Gen_List(i).Status.eq.On ) then
          type *,(i,'-->',Hour(Max_Hour).State(Min_State).
     .      Gen_List(i).Generation)
        endif
      enddo

      do Hour_Cnt = Max_Hour , 2 , -1
        Min_State = Hour(Hour_Cnt).State(Min_State).Path
        Min_Cost = Hour(Hour_Cnt-1).State(Min_State).Cost
          type * , ('Hour : ' , Hour_Cnt-1 , ' ' ,
     .              'State : ' , Min_State , ' ' ,
     .              'Cost : ' , Min_Cost          )
        do i = 1 , Max_Gen_No
          if ( Hour(Hour_Cnt-1).State(Min_State).
     .      Gen_List(i).Status.eq.On ) then
            type *,(i,'-->',Hour(Hour_Cnt-1).
     .        State(Min_State).Gen_List(i).Generation)
          endif
        enddo
      enddo
!-------------------------------------------------------------
      end
```

```
      Options / Extend_Source


      Program UCSUB
!-----------------------------------------------------------------
      Implicit Integer ( A - Z )
      Include 'UC_Parameter.for'
      Include '($JPIDEF)'
!-----------------------------------------------------------------
      integer Addr(2)
      integer Ret_Addr(2)
!-----------------------------------------------------------------
      character OutStr*10

      real Min_Load
      real PcostVal
      real ScostVal
      real Cost
      real Min_Cost

      real Pcost
      real Scost
      real P_List(1:Max_State_No,1:Max_Gen_No)

      integer Status
      integer BinNum(1:20)
      integer Max_Gen
      integer L
      integer State_Cnt
      integer i,j
      integer Min_State
      integer Max_Inc
      integer Gen_No
      integer BinNo
      integer No
      integer Hour_No
      integer State_no
      integer NumOfCycPeak
      integer OnOff_List(1:Max_State_No,1:Max_Gen_No)

      character*4 OutFile(5) / 'Out1' , 'Out2' , 'Out3' ,
     .                         'Out4' , 'Out5' /
!-----------------------------------------------------------------
      integer Hour_Cnt
      character Hour_Cnt_Eqv*4
      Equivalence ( Hour_Cnt , Hour_Cnt_Eqv )
!-----------------------------------------------------------------
      integer Process_No
      character Process_No_Eqv*4
      Equivalence ( Process_No , Process_No_Eqv )
!-----------------------------------------------------------------
      integer MaxStateSect_Prev
```

```
        character MaxStateSect_Prev_Eqv*4
        Equivalence ( MaxStateSect_Prev ,
     .                MaxStateSect_Prev_Eqv )
!-------------------------------------------------------------
        integer Start_Sect
        integer End_Sect
        character Start_Sect_Eqv*4
        character End_Sect_Eqv*4
        Equivalence ( Start_Sect , Start_Sect_Eqv )
        Equivalence ( End_Sect , End_Sect_Eqv )
!-------------------------------------------------------------
        integer Max_State
        character Max_State_Eqv*4
        Equivalence ( Max_State , Max_State_Eqv )
!-------------------------------------------------------------
        integer Start
        integer End
        character Start_Eqv*4
        character End_Eqv*4
        Equivalence ( Start , Start_Eqv )
        Equivalence ( End , End_Eqv )
!-------------------------------------------------------------
        integer Flag_Read
        integer Flag_Write
        character Flag_Read_Eqv*4
        character Flag_Write_Eqv*4
        Equivalence ( Flag_Read , Flag_Read_Eqv )
        Equivalence ( Flag_Write , Flag_Write_Eqv )
!-------------------------------------------------------------
        real Load
        character*4 Load_Eqv
        Equivalence ( Load , Load_Eqv )
!-------------------------------------------------------------
        Record / Hour_Rec / State
        Record / Hour_Rec / StatePrev
!-------------------------------------------------------------
        character*(PageFile_Len) Page_Data
        Common / Page_Common / Page_Data
!-------------------------------------------------------------
        Parameter GenList_Start = 65
        Parameter GenList_End = GenList_Start +
     .    GenListFull_Char - 1
!-------------------------------------------------------------
        call LIB$Init_Timer
!-------------------------------------------------------------
        Addr(1) = %LOC(Page_Data(1:1))
        Addr(2) = %LOC(Page_Data(PageFile_Len:PageFile_Len))
!-------------------------------------------------------------
        call Access_Page ( 'COMM' , Addr , Ret_Addr )
!-------------------------------------------------------------
        Status = SYS$ASCEFC ( %VAL(Base_Flag) , 'Cluster',,)
        if ( .NOT.Status ) call LIB$Signal ( %VAL(Status) )
```

```
!-----------------------------------------------------------------
      Status = SYS$WAITFR ( %VAL(base_Flag+ReadFlag) )
      if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
      Status = SYS$CLREF ( %VAL(base_Flag+ReadFlag) )
      if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )

      Hour_Cnt_Eqv =  Page_Data(1:4)
      MaxArea_Eqv = Page_Data(5:8)
      MaxGenListFull_Eqv = Page_Data(9:12)
      PFixed_Eqv = Page_Data(13:16)
      Max_Load_Eqv = Page_Data(17:20)
      Load_Eqv = Page_Data(21:24)
      Max_State_Eqv = Page_Data(25:28)
      MaxStateSect_Prev_Eqv = Page_Data(29:32)
      Start_Eqv = Page_Data(33:36)
      End_Eqv = Page_Data(37:40)
      Start_Sect_Eqv = Page_Data(41:44)
      End_Sect_Eqv = Page_Data(45:48)
      Flag_Read_Eqv = Page_Data(49:52)
      Flag_Write_Eqv = Page_Data(53:56)
      Process_No_Eqv = Page_Data(57:60)
      PfixedCost_Eqv = Page_Data(61:64)
      GenListFull_Eqv =
     .    Page_Data(GenList_Start:GenList_End)

      Status = SYS$SETEF ( %VAL(base_Flag+WriteFlag) )
      if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )

      Status = SYS$WAITFR ( %VAL(base_Flag+ReadFlag) )
      if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
      Status = SYS$CLREF ( %VAL(base_Flag+ReadFlag) )
      if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
      Area.Area_Eqv = Page_Data(1:Area_len)
      Status = SYS$SETEF ( %VAL(base_Flag+WriteFlag) )
      if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )

      Status = SYS$WAITFR ( %VAL(base_Flag+ReadFlag) )
      if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
      Status = SYS$CLREF ( %VAL(base_Flag+ReadFlag) )
      if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
      Plant.Plant_Eqv = Page_Data(1:Plant_len)
      Status = SYS$SETEF ( %VAL(base_Flag+WriteFlag) )
      if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )

      do i = 1 , Gen_Arr
        Status = SYS$WAITFR ( %VAL(base_Flag+ReadFlag) )
        if ( .Not.Status ) call LIB$Signal ( %VAL(Status))
        Status = SYS$CLREF ( %VAL(base_Flag+ReadFlag) )
        if ( .Not.Status ) call LIB$Signal ( %VAL(Status))
        Gen.Gen_Eqv(i) = Page_Data(1:Gen_Char)
        Status = SYS$SETEF ( %VAL(base_Flag+WriteFlag) )
        if ( .Not.Status ) call LIB$Signal ( %VAL(Status))
      enddo
```

```
do i = 1 , GenListOn_Arr
  Status = SYS$WAITFR ( %VAL(base_Flag+ReadFlag) )
  if ( .Not.Status ) call LIB$Signal ( %VAL(Status))
  Status = SYS$CLREF ( %VAL(base_Flag+ReadFlag) )
  if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
  GenListOn_Eqv(i) = Page_Data(1:GenListOn_Char)
  Status = SYS$SETEF ( %VAL(base_Flag+WriteFlag) )
  if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
enddo

do i = 1 , MaxGenListOn_Arr
  Status = SYS$WAITFR ( %VAL(base_Flag+ReadFlag) )
  if ( .Not.Status ) call LIB$Signal ( %VAL(Status))
  Status = SYS$CLREF ( %VAL(base_Flag+ReadFlag) )
  if ( .Not.Status ) call LIB$Signal ( %VAL(Status))
  MaxGenListOn_Eqv(i)=Page_Data(1:MaxGenListOn_Char)
  Status = SYS$SETEF ( %VAL(base_Flag+WriteFlag) )
  if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
enddo

do i = 1 , ReserveReq_Arr
  Status = SYS$WAITFR ( %VAL(base_Flag+ReadFlag) )
  if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
  Status = SYS$CLREF ( %VAL(base_Flag+ReadFlag) )
  if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
  ReserveReq_Eqv(i) = Page_Data(1:ReserveReq_Char)
  Status = SYS$SETEF ( %VAL(base_Flag+WriteFlag) )
  if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
enddo

do i = 1 , ReadyReserveReq_Arr
  Status = SYS$WAITFR ( %VAL(base_Flag+ReadFlag) )
  if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
  Status = SYS$CLREF ( %VAL(base_Flag+ReadFlag) )
  if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
   ReadyReserveReq_Eqv(i) =
     Page_Data(1:ReadyReserveReq_Char)
  Status = SYS$SETEF ( %VAL(base_Flag+WriteFlag) )
  if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
enddo

do i = 1 , FastP_Arr
  Status = SYS$WAITFR ( %VAL(base_Flag+readFlag) )
  if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
  Status = SYS$CLREF ( %VAL(base_Flag+ReadFlag) )
  if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
  FastP_Eqv(i) = Page_Data(1:FastP_Char)
  Status = SYS$SETEF ( %VAL(base_Flag+WriteFlag) )
  if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
enddo
```

```
      do i = 1 , MaxStateSect_Prev
        Status = SYS$WAITFR ( %VAL(base_Flag+readFlag) )
        if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
        Status = SYS$CLREF ( %VAL(base_Flag+ReadFlag) )
        if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
        StatePrev.StateRec_Eqv(i) =
     .    Page_Data(1:StateRec_Char)
        Status = SYS$SETEF ( %VAL(base_Flag+WriteFlag) )
        if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
      enddo

      do i = Start_Sect , End_Sect
        Start_Char = (Start-1)*Total_State_Len+1
        End_Char = End*Total_State_Len
        if ( Start_Char.lt.(i-1)*StateRec_Char+1 .or.
     .       Start_Char.ge.i*StateRec_Char          ) then
          Start_Char = 1
        else
          Start_Char = JMOD ( Start_Char , StateRec_Char )
        endif
        if ( End_Char.lt.(i-1)*StateRec_Char+1 .or.
     .       End_Char.ge.i*StateRec_Char          ) then
          End_Char = StateRec_Char
        else
          End_Char = JMOD ( End_Char , StateRec_Char )
        endif
        Status = SYS$WAITFR ( %VAL(base_Flag+ReadFlag) )
        if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
        Status = SYS$CLREF ( %VAL(base_Flag+ReadFlag) )
        if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
        State.StateRec_Eqv(i)(Start_Char:End_Char) =
     .    Page_Data(Start_Char:End_Char)
        Status = SYS$SETEF ( %VAL(base_Flag+WriteFlag) )
        if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
      enddo
!-----------------------------------------------------------------
      call LIB$INIT_TIMER
!-----------------------------------------------------------------
      Nosolution = .True.
      do State_Cnt = Start , End
        Min_State = 0
        Min_Cost = Max_Real
        do L = 1 , Max_State
          PCostVal = Pcost ( State.State(State_Cnt) ,
     .                       StatePrev.State(L) ,
     .                       State_Cnt       ,
     .                       L               ,
     .                       P_List          ,
     .                       OnOff_List      ,
     .                       Hour_Cnt        ,
     .                       Load            )
```

```
            ScostVal = Scost ( State.State(State_Cnt) ,
     .                         StatePrev.State(L)     )

            Cost = StatePrev.State(L).Cost + PcostVal +
     .             SCostVal

            if ( Cost.lt.Min_Cost ) then
              Min_Cost = Cost
              Min_State = L
            endif
          enddo ! L
!-----------------------------------------------------------------
          if ( Min_State.ne.0 ) then
            Nosolution = .False.
          endif
!-----------------------------------------------------------------
          State.State(State_Cnt).Path = Min_State
          State.State(State_Cnt).Cost = Min_Cost+PFixedCost
!-----------------------------------------------------------------
          if ( Min_State.ne.0) then
              do i = 1 , MaxGenlistFull
              Gen_No = GenListFull(i)
              State.State(State_Cnt).
     .          Gen_List(Gen_No).Generation =
     .          P_List(Min_State,Gen_No)
              State.State(State_Cnt).
     .          Gen_List(Gen_No).OnOffTime =
     .          OnOff_List(Min_State,Gen_No)
            enddo
          endif
!-----------------------------------------------------------------
        enddo ! state_cnt
!-----------------------------------------------------------------
9999      Continue
!-----------------------------------------------------------------
        call LIB$SHOW_TIMER
!-----------------------------------------------------------------
        Status = SYS$SETEF ( %VAL(Base_Flag+Flag_Write) )
        if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
        Status = SYS$WAITFR ( %VAL(Base_Flag+Flag_Read) )
        if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
!-----------------------------------------------------------------
        do i = Start_Sect , End_Sect
          Start_Char = (Start-1)*Total_State_Len+1
          End_Char = End*Total_State_Len
          if ( Start_Char.lt.(i-1)*StateRec_Char+1 .or.
     .           Start_Char.ge.i*StateRec_Char          ) then
            Start_Char = 1
          else
            Start_Char = JMOD ( Start_Char , StateRec_Char )
          endif
          if ( End_Char.lt.(i-1)*StateRec_Char+1 .or.
```

```
     .          End_Char.ge.i*StateRec_Char          ) then
              End_Char = StateRec_Char
            else
              End_Char = JMOD ( End_Char , StateRec_Char )
            endif

            Page_Data(Start_Char:End_Char) =
     .        State.StateRec_Eqv(i)(Start_Char:End_Char)
            Status = SYS$SETEF ( %VAL(base_Flag+ReadFlag) )
            if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
            Status = SYS$WAITFR ( %VAL(base_Flag+WriteFlag) )
            if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
            Status = SYS$CLREF ( %VAL(base_Flag+WriteFlag) )
            if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
          enddo
!-------------------------------------------------------------
          Page_Data(1:4) = NoSolution_Eqv
          Status = SYS$SETEF ( %VAL(base_Flag+ReadFlag) )
          if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
!-------------------------------------------------------------
          call Close_Page ( Ret_Addr )
!-------------------------------------------------------------
          Status = SYS$SETEF ( %VAL(base_Flag+Flag_Write) )
          if ( .Not.Status ) call LIB$Signal ( %VAL(Status) )
!-------------------------------------------------------------
          end
```

```
!-----------------------------------------------------------------
! UC_PARAMETER.FOR
!-----------------------------------------------------------------
      Parameter Max_Area_Limit = 10
      Parameter Max_Plant_Limit = 10
      Parameter Max_Real = 99999999999.9
      Parameter MaxInt = 99999999
      Parameter Max_Gen_No = 13
      Parameter Max_State_No = 32 ! 2^Max_Gen_No-1
      Parameter Max_Inc_Val = 500
      Parameter Max_Hour = 12
      Parameter On = 1
      Parameter Off = 0
      Parameter MustRun = 0
      Parameter Peaker = 1
      Parameter Cycling = 2
      Parameter FixedMW = 3
      Parameter Avalaible = 0
      Parameter UnAvalaible = 1
      Parameter Inc_Val = 10.0
      Parameter MaxRampHour = 7
!-----------------------------------------------------------------
      Parameter PageFile_Len = 2000
      Parameter MaxProcessNo = 2
      Parameter Max_State_N = 4
!-----------------------------------------------------------------
      Parameter Base_Flag = 64

      Parameter Flag_Read1 = 0
      Parameter Flag_Read2 = 1
      Parameter Flag_Read3 = 2
      Parameter Flag_Read4 = 3
      Parameter Flag_Read5 = 4
      Parameter Flag_Read6 = 5
      Parameter Flag_Read7 = 6
      Parameter Flag_Read8 = 7
      Parameter Flag_Read9 = 8
      Parameter Flag_Read10 = 9

      Parameter Flag_Write1 = 10
      Parameter Flag_Write2 = 11
      Parameter Flag_Write3 = 12
      Parameter Flag_Write4 = 13
      Parameter Flag_Write5 = 14
      Parameter Flag_Write6 = 15
      Parameter Flag_Write7 = 16
      Parameter Flag_Write8 = 17
      Parameter Flag_Write9 = 18
      Parameter Flag_Write10 = 19

      Parameter OK_Flag = 20
      Parameter WriteFlag = 21
```

```
      Parameter ReadFlag = 22
!-----------------------------------------------------------------
      Structure / P_MIN_MAX /
        integer P_Min
        integer P_Max
      End Structure
!-----------------------------------------------------------------
      Parameter Area_len = Max_Area_Limit*(Max_Gen_No*4+4*5)

      Structure / Area_Rec /
        integer Gen_List(1:Max_Gen_No)
        integer MaxList
        real ReserveReq
        real ReadyReserveReq
        real ReserveLimit
        real ReadyReserveLimit
      End Structure

      Structure / Area_Rec1 /
        Union
          Map
            Record / Area_rec / A(1:Max_Area_Limit)
          End Map
          Map
            Character*(Area_Len) Area_Eqv
          End Map
        End Union
      End Structure
!-----------------------------------------------------------------
      Parameter Plant_len = Max_Plant_Limit*(Max_Gen_No*4+4)

      Structure / Plant_Rec /
        integer Gen_List(1:Max_Gen_No)
        integer MaxList
      End Structure

      Structure / Plant_Rec1 /
        Union
          Map
            Record / Plant_rec / P(1:Max_Plant_Limit)
          End Map
          Map
             Character*(Plant_Len) Plant_Eqv
          End Map
        End Union
      End Structure
!-----------------------------------------------------------------
      Parameter Total_State_Len = Max_Gen_No*12 + 8
      Parameter Total_StateRec_Len = (Max_State_No+1)*
     .  Total_State_Len
      Parameter StateRec_Arr =
     .  JINT (Total_StateRec_Len*1.0/PageFile_Len)+1
```

```
      Parameter StateRec_Char =
     .  JINT (Total_StateRec_Len*1.0/StateRec_Arr)+1

      Structure / Gen_List_Rec /
        integer Status
        integer OnOffTime
        real Generation
      End Structure

      Structure / State_Rec /
        Record / Gen_List_Rec / Gen_List (1:Max_Gen_No)
        real Cost
        real Path
      End Structure

      Structure / Hour_Rec /
        Union
          Map
            Record / State_Rec / State(1:Max_State_No)
          End Map
          Map
             character*(StateRec_Char) StateRec_Eqv
     .                                 (1:StateRec_Arr)
          End Map
        End Union
        Union
          Map
            integer Max_State
          End Map
          Map
            character*4 Max_State_Eqv
          End Map
        End Union
      End Structure
!-------------------------------------------------------------
      Parameter Total_Gen_Len=Max_Gen_No*(max_Hour*4*2+23*4)
      Parameter Gen_Arr=JINT(Total_Gen_Len*1.0/PageFile_Len)+1
      Parameter Gen_Char=JINT(Total_Gen_Len*1.0/Gen_Arr)+1
```

```
      Parameter Total_GenListOn_Len = Max_Hour*
     .   Max_State_No*Max_Gen_No*4
      Parameter GenListOn_Arr =
     .   JINT (Total_GenListOn_Len*1.0/PageFile_Len)+1
      Parameter GenListOn_Char =
     .   JINT (Total_GenListOn_Len*1.0/GenListOn_Arr)+1

      integer GenListOn(1:Max_State_No,1:Max_Hour,
     .   1:Max_Gen_No)
      character*(GenListOn_Char) GenListOn_Eqv
     .                           (1:GenListOn_Arr)
      Equivalence ( GenListOn , GenListOn_Eqv )
!-----------------------------------------------------------------
      Parameter GenListFull_Char = Max_Gen_No*4
      integer GenListFull(1:Max_Gen_No)
      character*(GenListFull_Char) GenListFull_Eqv
      Equivalence ( GenListFull , GenListFull_Eqv )
!-----------------------------------------------------------------
      Parameter Total_FastP_Len = Max_Hour*Max_State_No*4
      Parameter FastP_Arr = JINT (Total_FastP_Len*1.0/
     .    PageFile_Len)+1
      Parameter FastP_Char = JINT (Total_FastP_Len*1.0/
     .    FastP_Arr)+1

      real FastP(1:Max_State_No,1:Max_Hour)
      character*(FastP_Char) FastP_Eqv(1:FastP_Arr)
      Equivalence ( FastP , FastP_Eqv )
!-----------------------------------------------------------------
      Parameter Total_ReserveReq_Len=Max_Hour*Max_State_No*4
      Parameter ReserveReq_Arr =
     .   JINT (Total_ReserveReq_Len*1.0/PageFile_Len)+1
      Parameter ReserveReq_Char =
     .   JINT (Total_ReserveReq_Len*1.0/ReserveReq_Arr)+1

      real ReserveReq(1:Max_State_No,1:Max_Hour)
      character*(ReserveReq_Char) ReserveReq_Eqv
     .    (1:ReserveReq_Arr)
      Equivalence ( ReserveReq , ReserveReq_Eqv )
!-----------------------------------------------------------------
      Parameter Total_ReadyReserveReq_Len = Max_Hour*
     .    Max_State_No*4
      Paerameter ReadyReserveReq_Arr =
     .    JINT (Total_ReadyReserveReq_Len*1.0/PageFile_Len)+1
      Parameter ReadyReserveReq_Char =
     .    JINT (Total_ReadyReserveReq_Len*1.0/
     .     ReadyReserveReq_Arr)+1

      real ReadyReserveReq(1:Max_State_No,1:Max_Hour)
      character*(ReadyReserveReq_Char) ReadyReserveReq_Eqv
     .    (1:ReadyReserveReq_Arr)
      Equivalence ( ReadyReserveReq , ReadyReserveReq_Eqv )
!-----------------------------------------------------------------
```

```
        integer MaxArea
       character MaxArea_Eqv*4
       Equivalence ( MaxArea, MaxArea_Eqv )
!-------------------------------------------------------------
       integer MaxGenListFull
       character MaxGenListFull_Eqv*4
       Equivalence ( MaxGenListFull_Eqv , MaxGenListFull )
!-------------------------------------------------------------
       integer MaxPlant
       character MaxPlant_Eqv*4
       Equivalence ( MaxPlant , MaxPlant_Eqv )
!-------------------------------------------------------------
       real Pfixed
       character PFixed_Eqv*4
       Equivalence ( PFixed , Pfixed_Eqv )
       real PfixedCost
       character*4 PfixedCost_Eqv
       Equivalence ( PfixedCost , PfixedCost_Eqv )
       real Max_Load
       character Max_Load_Eqv*4
       Equivalence ( Max_Load_Eqv , Max_Load )
!-------------------------------------------------------------
       logical NoSolution
       character NoSolution_Eqv*4
       Equivalence ( NoSolution_Eqv,NoSolution )
       Record / Plant_Rec1 / Plant
       Record / Area_Rec1 / Area
       Record / Gen_Rec1 / Gen
       integer ReadFlagList(5) / Flag_Read1 ,
     .                           Flag_Read2 ,
     .                           Flag_Read3 ,
     .                           Flag_Read4 ,
     .                           Flag_Read5 /
       integer WriteFlagList(5) / Flag_Write1 ,
     .                            Flag_Write2 ,
     .                            Flag_Write3 ,
     .                            Flag_Write4 ,
     .                            Flag_Write5 /
       Common / UC_Common / Area              ,
     .                      MaxArea           ,
     .                      Plant             ,
     .                      MaxPlant          ,
     .                      GenListFull       ,
     .                      MaxGenListFull    ,
     .                      GenListOn         ,
     .                      MaxGenListOn      ,
     .                      Gen               ,
     .                      ReserveReq        ,
     .                      ReadyReserveReq   ,
     .                      FastP             ,
     .                      PFixed            ,
     .                      Max_Load
```


T. C.
Yükseköğretim Kurulu