The Pennsylvania State University

The Graduate School

College of Engineering

UNIT COMMITMENT PROBLEMS: A COMPARISON OF ALGORITHMS AND HEURISTICS

A Thesis in

Industrial Engineering and Operations Research

by

Fan You

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

December 2015

The thesis of Fan You was reviewed and approved* by the following:

Mort D. Webster
Associate Professor of Energy Engineering
Thesis Co-Advisor

Jose A. Ventura
Professor of Industrial Engineering
Thesis Co-Advisor

Janis P. Terpenny
Professor of Industrial Engineering
Peter and Angela Dal Pezzo Department Head

*Signatures are on file in the Graduate School.

Abstract

Unit commitment problems have been an area of research interest for many decades. The general unit commitment problem is a model used to determine when to utilize different energy generation resources in order to minimize the cost of electricity generation while satisfying demand under the presence of a variety of resource constraints. In this work, we compare variant models and algorithms for solving the unit commitment problem. We measure each model by comparing runtimes across a variety of time horizons with a selection of different scenarios.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Unit Commitment Problems

1.1.1 Definition of the Unit Commitment Problem

The Unit Commitment (UC) problem can be stated as finding the optimal scheduling of production of electric power generating units over a short term, typically from 24 hours to one week, in order to minimize the operations cost. This optimal solution must satisfy the operating constraints and must satisfy demand [1].

A literature review of unit commitment problems was provided in [2]. Traditionally the unit commitment problem was formulated as a mixed-integer program [3, 4]. Branch and Bound algorithms have been developed in order to solve the unit commitment problem [5], and the Lagrangian Relaxation optimization technique is also widely used to solve the unit commitment problem [6, 7].

1.1.2 Motivation of the Thesis

The UC problem is a mixed-integer programming problem and is in the class of NP-hard problems [8]. Because of its size and NP-hardness, solving the UC problem and obtain the optimal solution is computationally expensive [9]. When integrated generation expansion and unit commitment models are used to determine what generating units should be constructed and when generating units should come online over a long-term planning horizon, a large number of UC problems need to be solved and methods with faster convergence rate would greatly improve the performance of these models. Therefore several heuristic methods are developed in order to solve the UC problems with significantly reduced computational effort.

1.2 Dynamic Programming and Markov Decision Processes

The optimization of problems over time and under uncertainty arises in many settings, these problems involve making decisions, then observing information, after which we make more decisions and then obtain more information, and so on, know as sequential decision problems. The fields of operations research and artificial intelligence work primarily with discrete states and decisions, modeled in discrete time, and these problems are studied under the umbrella of "Markov decision processes". Markov decision processes (MDPs) provide a mathematical framework for modeling decision making in situations where the outcomes are partly random and partly under the control of a decision maker [10]. MDPs are frequently solved using Dynamic Programming (DP) [11].

Dynamic Programming is a class of optimization problems designed to aid decision-making in multiple stages. In such situations, decisions made in the current stage to lower the current cost may raise future stages' expected costs, and DP captures such trade-offs [12].The two prerequisites in DP are (1) a discrete-time dynamic system and (2) a cost function that is additive over time, and depends on the states and controls [13].

To mathematically formulate the MDP, a very elegant approach is provided in [10]. Assume that we have a discrete state space $\mathcal{S} = (1, 2, ..., |\mathcal{S}|)$. Next assume that there is a set of decisions or actions, which we denote by $a \in \mathcal{A}$, and that we can compute a contribution given by $C(s, a)$.

Finally, assume that we are given a transition matrix $p_t(S_{t+1}|S_t, a_t)$ that gives the probability that if we are in state S_t (at time t) and take action a_t, then we will next be in state S_{t+1}.

1.2.1 Finite Horizon Dynamic Programming

Given these assumptions, the MDP can be solve via the following optimality equation, with finite state space, maximizing the finite horizon contribution:

$$\max_{\pi \in \Pi} \quad \mathbb{E} \sum_{t=0}^{T} \gamma^t C_t(S_t, X^\pi(S_t)), \tag{1.1}$$

where

- t is the index of time stages, $t = 1, 2, ..., T$;
- S_t is the state variable at stage t;
- π is a policy (decision rule), belonging to a set of policies Π;
- $X^\pi(S_t)$ is a function of state S_t, and represents the decision made at S_t under policy π;
- $C_t(S_t, X^\pi(S_t))$ is a function of state S_t and decision $X^\pi(S_t)$ and represents the reward from choosing decision $X^\pi(S_t)$ at state S_t;
- γ^t is the discount factor at stage t, $0 < \gamma^t \leq 1$ ($\gamma^t < 1$ means that the future reward at stage t has a lower value compared with the same reward obtained at current stage [13]).

The key idea in DP is that we do not have to solve the entire problem at once. Suppose that we are solving a deterministic problem, if we are in state $S_t = i$,and take action $a_t = j$, our transition function will tell us that we are going to land in some state $S_{t+1} = S^M(S_t, a_t)$. Suppose we have a function $V_{t+1}(S_{t+1})$ that tells us the value of being in state S_{t+1}, then we could evaluate each possible action a_t and simply choose the action a_t that has the largest one-period contribution, $C_t(S_t, a_t)$, plus the value of landing in state $S_{t+1} = S^M(S_t, a_t)$, which we represent using $V_{t+1}(S_{t+1})$. Since this value represents the contribution we receive in one time period in

the future, we discount this by a factor γ. In other words, we have to solve

$$a_t^*(S_t) = \arg\max_{a_t \in A_t} (C_t(S_t, a_t) + \gamma V_{t+1}(S_{t+1})), \tag{1.2}$$

where "arg max" means that we want to choose the action a_t that maximizes the expression in the parentheses. We also note that $S_{t+1}$is a function of S_t and a_t, meaning that we could write it as $S_{t+1}(S_t, a_t)$. The value of being in state S_t is the value of choosing the optimal decision $a_t^*(S_t)$. That is,

$$\begin{aligned} V_t(S_t) &= \max_{a_t \in A_t} (C_t(S_t, a_t) + \gamma V_{t+1}(S_{t+1}(S_t, a_t))) \\ &= C_t(S_t, a_t^*(S_t)) + \gamma V_{t+1}(S_{t+1}(S_t, a_t^*(S_t))). \end{aligned} \tag{1.3}$$

Equation (1.3) is the optimality equation for deterministic problems.

When we encounter a finite horizon problem, we assume that we are given the function $V_T(S_T)$ as data. Solving a finite horizon problem, in principle, is straightforward. As outlined in Algorithm 1, we simply have to start at the last time period, compute the value function for each possible state $s \in \mathcal{S}$, and then step back another time period. This way at time period t we have already computed $V_{t+1}(S)$. Consequently, this method is often referred to as "backward dynamic programming." The critical element that attracts so much attention is the requirement that we compute the value function $V_t(S_t)$ for all states $S_t \in \mathcal{S}$.

Algorithm 1 Backward Dynamic Programming

Step 0.

Initialization:

Initialize the terminal contribution $V_T(S_T)$.

Set $t = T - 1$.

Step 1.

Calculate:

$$V_t(S_t) = \max_{a_t} \{C_t(S_t, a_t) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|S_t, a_t) V_{t+1}(S')\} \tag{1.4}$$

for all $S_t \in \mathcal{S}$.

Step 2.

If $t > 0$,decrement t and return to step 1. Else stop.

1.2.2 Infinite Horizon Dynamic Programming

Typically we use infinite horizon fomrulations when we wish to study a problem where the parameters of the contribution function, transition function, and the process governing the exogenous information process do not vary over time, although they might vary in cycles. We wish to study such problems in steady state. Moreover, infinite horizon problems provide a number of insights into the properties of the problems and algorithms, leading to an elegant theory that has evolved around this problem class.

The finite horizon optimality equation can be given as:

$$V_t(S_t) = C_t(S_t, a_t^*(S_t)) + \gamma V_{t+1}(S_{t+1}(S_t, a_t^*(S_t))). \tag{1.5}$$

We can think of a steady-state problem as one without the time dimension, thus letting $V(s) = \lim_{t\to\infty} V_t(S_t)$, we obtain the steady-state optimality equations:

$$V(s) = \max_{a\in\mathcal{A}}\{C(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathbb{P}(s'|s,a)V(s')\}. \tag{1.6}$$

There are several algorithmic strategies for solving infinite horizon problems. The first, *value iteration* [10], is the most widely used method. It involves iteratively estimating the value function. At each iteration, the estimate of the value function determines which decisions we will make and as a result defines a policy. The second strategy is *policy iteration* [10]. At every iteration, we define a policy (the rule for determine decisions) and then determine the value function for that policy. Careful examination of value and policy iteration reveals that these are closely related strategies that can be viewed as special cases of a general strategy that uses value and policy iteration. Finally, the third major algorithmic strategy exploits the observation that the value function can be viewed as the solution to a specially structured linear programming problem.

1.3 Principal Components Analysis and Regression

1.3.1 Definition of Principal Components

The central idea of principal component analysis (PCA) is to reduce the dimensionality of a data set which consists of a large number of interrelated variables, while retaining as much as possible of the variation present in the data set. This is achieved by transforming to a new set of variables, the principal components (PCs), which are uncorrelated, and which are ordered so that the first *few* retain most of the of the variation present in *all* of the original variables [14].

Suppose that x is a vector of p random variables, and the variances of the p random variables and the structure of the covariances or correlations between the p variables are of interest. We could look for a few ($\ll p$) derived variables which preserve most of the information given by these variances and correlations or covariances.

Although PCA does not ignore covariances and correlations, it concentrates on variances. The first step is to look for a linear function $\alpha_1' x$ of the elements of x which has maximum variance, where α_1 is a vector of p constants, $\alpha_{11}, \alpha_{12}, ..., \alpha_{1p}$, so that

$$\alpha_1' x = \alpha_{11} x_1 + \alpha_{12} x_2 + \cdots + \alpha_{1p} x_{1p} = \sum_{j=1}^{p} \alpha_{1j} x_j. \tag{1.7}$$

Next, look for a linear function $\alpha_2' x$, uncorrelated with $\alpha_1' x$, which has maximum variances, and so on, so that at the kth stage a linear function $\alpha_k' x$ is found which has maximum variance subject to being uncorrelated with $\alpha_1' x$,$\alpha_2' x$,...,$\alpha_k - 1' x$. The kth derived variable, $\alpha_k' x$, is called the kth Principal Components(PC). Up to p PCs could be found, but the dimensionality reduction is achieved when most of the variation in x will be accounted for by m PCs, where $m \ll p$. The reduction in dimensionality is demonstrated by unrealistic, but simple, case where $p = 2$.

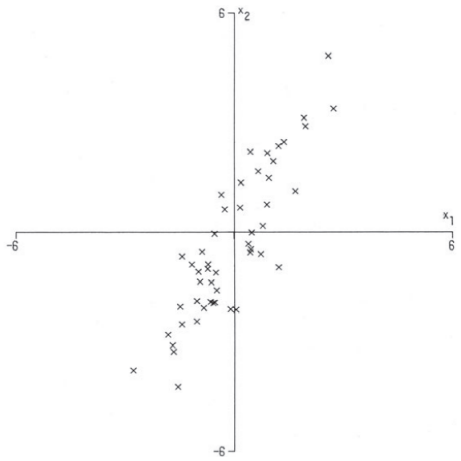


Figure 1.1. Plot of 50 observations on x_1, x_2

.

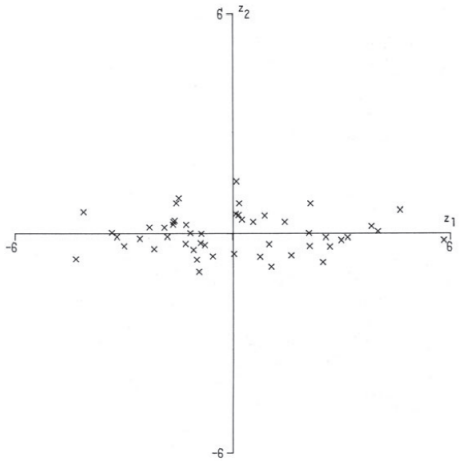


Figure 1.2. Plot of 50 observations on PCs z_1, z_2

.

Figure 1.1 [14] gives a plot of 50 observations on two highly correlated variables x_1, x_2. If we transform to PCs z_1, z_2, we obtain the plot given in Figure 1.2 [14]. It is clear that there is greater variation in the direction of z_1 than in either of the original variables, but much less variation in the direction of z_2.

1.3.2 Derivation of Principal Components

Having defined PCs, we need to know how to find them. Consider the case where the vector of random variable x has a known covariance matrix Σ. It turns out that, for $k = 1, 2, ..., p$, the k PC is given by $z_k = \alpha_k' x$ where α_k is an eigenvector of Σ corresponding to its k largest eigenvalue λ_k. Furthermore, if α_k is chosen to have unit length, then $var(z_k) = \lambda_k$, where $var(z_k)$ denotes the variances of z_k.

A standard derivation of the above result is given in [14].To derive the form of the PCs, consider the first $\alpha_1' x$,α_1 maximizes $var[\alpha_1' x] = \alpha_1' \Sigma \alpha_1$. The maximum will not be achieved for finite α_1, so a normalization constraint must be imposed. The most convenient constraint here is $\alpha_1' \alpha_1 = 1$. To maximize $\alpha_1' \Sigma \alpha_1$ subject to $\alpha_1' \alpha_1 = 1$, use the technique of Lagrange multipliers and maximize

$$\alpha_1' \Sigma \alpha_1 - \lambda(\alpha_1' \alpha_1 - 1), \tag{1.8}$$

where λ is a Lagrange multiplier. Differentiation with respect to α_1 gives

$$(\Sigma - \lambda I_p)\alpha_1 = 0, \tag{1.9}$$

where I_p is the identity matrix. Thus, λ is an eigenvalue of Σ and α_1 is the corresponding eigenvector. To decide which of the p eigenvectors is the maximizing value of α_1, note that the quantity to be maximized is

$$\alpha_1' \Sigma \alpha_1 = \alpha_1' \lambda \alpha_1 = \lambda \alpha_1' \alpha_1 = \lambda, \tag{1.10}$$

so *lambda* must be as large as possible. Thus, α_1 is the eigenvector corresponding to the largest eigenvalue of Σ, and $var(\alpha_1') = \alpha_1' \Sigma \alpha_1 = \lambda_1$, the largest eigenvalue.

In general, the kth PC of x is $\alpha_k' x$and $var(\alpha_1' x) = \lambda_k$, where λ_k is the kth largest eigenvalue of Σ, and α_k is the corresponding eigenvector.

1.3.3 Principal Components Regression

In multiple regression, one of the major difficulties with the usual least squares estimators is the problem of multicollinearity. To overcome this problem, various approaches have been proposed. One possibility is to use only a subset of the predictor variables. The best-known approach, generally known as PC regression, simply starts by using PCs of the predictor variables in place of the predictor variables. Since the PCs are uncorrelated, the regression calculations are simplified. If all the PCs are included in the regression, then the resulting model is equivalent to that obtained by least squares. However if some of the PCs are deleted from the regression equation, estimators of the coefficients in the original regression equations are obtained which are usually biased, but which simultaneously greatly reduce any large variances for regression coefficient estimators which have been caused by multicollinearities.

Consider the standard regression model

$$y = X\beta + \epsilon, \tag{1.11}$$

where y is a vector of n observations on the dependent variable, measured about their mean, X is a $(n \times p)$ matrix whose (i, j)th element is the value of the jth predictor variable for the ith observation, again measured about its mean, β is a vector of regression coefficients and ϵ is the vector of error terms. The value of the PCs for each observation are given by

$$Z = XA, \tag{1.12}$$

where the (i, k)th element of Z is the value(score) of the kth PC for the ith observation, and A is a $p \times p$ matrix whose kth column is the kth eigenvector of $X'X$.

Because A is orthogonal, $X\beta$ can be rewritten as $XAA'\beta = Z\gamma$, where $\gamma = A'\beta$.Equation 2.17 can therefore be written as

$$y = Z\gamma + \epsilon, \tag{1.13}$$

which has simply replaced the predictor variables by their PCs in the regression model. Principal

component regression can be defined as the use of the model 1.13 or the reduced model

$$y = Z_m \gamma_m + \epsilon_m, \tag{1.14}$$

where γ_m is a vector of m elements which are a subset of elements of γ, Z_m is a $(n \times m)$ matrix whose columns are the corresponding subset of columns of Z, and ϵ_m is the appropriate error term.

1.4 General Regression Neural Networks

General Regression Neural Networks (GRNN) is a type of neural network proposed in [15]. GRNN has similar architecture to multi-layer perceptron neural networks, with a basic difference. Traditional networks perform classification when the target variable is categorical, whereas GRNNs perform perform regression where the target variable is continuous. GRNN hence is capable of estimating any arbitrary function from historical data.

Assume that $f(x, y)$ represents the known joint continuous probability density function of a vector random variable x, and a scalar random variable y. Let x_0 be a particular measured value of the random variable x. The regression of y on x_0 is given by

$$\hat{y}(x_0) = E[y|x_0] = \frac{\int_{-\infty}^{\infty} y f(x_0, y) dy}{\int_{-\infty}^{\infty} f(x_0, y) dy}. \tag{1.15}$$

When the density $f(x, y)$ is not known, it must usually be estimated from a sample of observations of x and y. Density function $f(x_0, y)$ can be obtained according to 1.15 by sample data collection $\{x_i, y_i\}_{i=1}^{n}$

$$f(x_0, y) = \frac{1}{n(2\pi)^{(p+1)/2} w_1 w_2 \cdots w_p w_y} \sum_{i=1}^{n} e^{-d(x_0, x_i)} e^{-d(y, y_i)} \tag{1.16}$$

Where

$$d(x_0, x_i) = \frac{(x_0 - x_i)^2}{2\sigma^2}, \qquad d(y, y_i) = \frac{(y - y_i)^2}{2\sigma^2} \tag{1.17}$$

and n is the number of sample observations, p is the dimension of x and σ is the spread factor of Gauss function (also called the smooth parameter).

By substituting the density function 1.16 into the conditional mean 1.15, and exchange sequence of integral and additive, $\hat{y}(x_0)$ can be obtained as

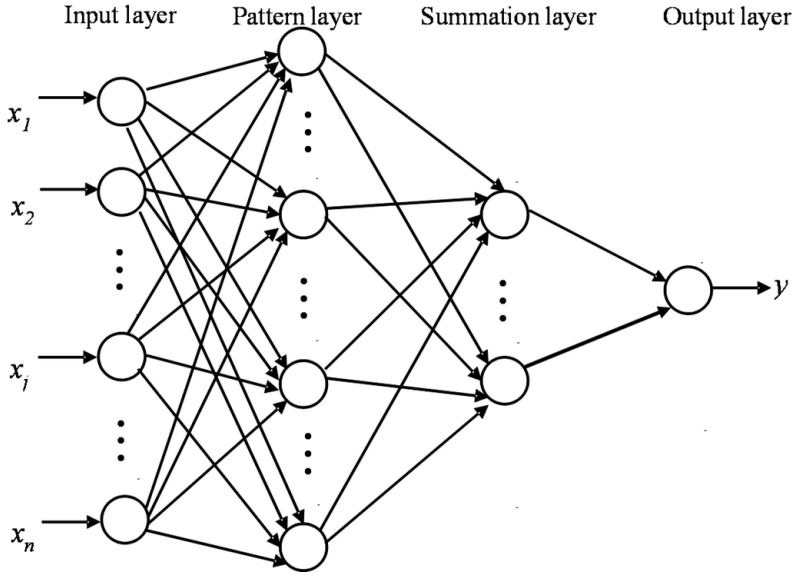
$$\hat{y}(x_0) = \frac{\sum_{i=1}^{n}(e^{-d(x_0,x_i)} \int_{-\infty}^{\infty} y e^{-d(y,y_i)} dy)}{\sum_{i=1}^{n}(e^{-d(x_0,x_i)} \int_{-\infty}^{\infty} e^{-d(y,y_i)} dy)}. \tag{1.18}$$

Performing the two integrals of 1.18 yields the following

$$\hat{y}(x_0) = \frac{\sum_{i=1}^{n} y_i e^{-d(x_0,x_i)}}{\sum_{i=1}^{n} e^{-d(x_0,x_i)}}. \tag{1.19}$$

So the predicted value $\hat{y}(x_0)$ is the weighted average of all of the observed values y_i where each observed value is weighted exponentially according to its Euclidean distance from x_0.

GRNN networks have four layers of processing units as shown in 1.3. Each layer of processing units is assigned with a specific computational function when non-linear regression is performed. These layers are the input layer, the pattern layer, the summation layer and the output layer.



[16]

Figure 1.3. GRNN architecture

.

The first layer of the network is responsible for the reception of information. This layer has as many neurons as there are input vectors. The input neurons feed the data to the second layer, the

number of neurons in this layer is equal to the number of cases in the training sets. The neurons of the third layer receive the outputs of the pattern neurons. This layer includes two units. The first unit calculates the sum of the weight of each unit's output from the second layer, the weight is the value of every training sample which can be regraded as shown in 1.19. Finally the fourth layer produces the estimated output component that is the localized average of the stored output patterns.

1.5 Organization of Thesis

This Thesis contains four chapters. Chapter 1 is the introductory chapter which provides motivation of the research and some background knowledge. Chapter 2 discusses different methods to solve the UC problem, both exact approaches and heuristics. Chapter 3 provides test examples for the methods discussed in Chapter 2, and the computational performance of the methods are evaluated. Chapter 4 is the concluding chapter, and provides the results of this Thesis and potential future research topics.

Chapter 2

Model Formulation and Solution Procedures

In this chapter, four approaches to formulate the UC problem are provided to solve the unit commitment problem. The first one uses mixed-integer programming. This method solves the UC problems by reducing the solution search space systematically through discarding the infeasible subsets [17]. The second method based on Dynamic Programming breaks large problems into smaller sub-problems by leveraging the optimal value of smaller problems to reduce the cost of solving the larger problem [18]. The third method relies on the priority list which is a pre-determined order of generating units. The fourth method is based on several predictive models which takes the economic dispatching solution of a subset of commitment states as input data to predict results of other dispatching problems.

2.1 Mixed-Integer Programming

Following the definition of the UC problem given in 1.1 and the model proposed in [1, 19], a deterministic mixed-integer programming model is given by,

$$\text{Minimize:} \quad \sum_{i=1}^{I}\sum_{t=1}^{T} z_{i,t}F_{i,t} + \sum_{i=1}^{I}\sum_{t=1}^{T} C_{i,t}(g_{i,t}) + \sum_{i=1}^{I}\sum_{t=1}^{T} y_{i,t}S_{i,t} \tag{2.1}$$

$$\text{subject to:} \quad \sum_{i=1}^{I} g_{i,t} >= d_t, t = 1, ..., T \tag{2.2}$$

$$g_{i,t} <= Q_i u_{i,t}, t = 1, ..., T, i = 1, ..., I \tag{2.3}$$

$$g_{i,t} >= q_i u_{i,t}, t = 1, ..., T, i = 1, ..., I \tag{2.4}$$

$$y_{i,t} >= u_{i,t} - u_{i,t-1}, t = 2, ..., T, i = 1, ..., I \tag{2.5}$$

$$z_{i,t} >= u_{i,t-1} - u_{i,t}, t = 2, ..., T, i = 1, ..., I \tag{2.6}$$

$$u_{i,t} - u_{i,t-1} <= u_{i,\tau}, \tau = t, ..., \min\{t + L_i - 1, T\}, t = 2, ..., T, i = 1, ..., I \tag{2.7}$$

$$u_{i,t-1} - u_{i,t} <= 1 - u_{i,\tau}, \tau = t, ..., \min\{t + L_i - 1, T\}, t = 2, ..., T, i = 1, ..., I \tag{2.8}$$

$$u_{i,t},\ y_{i,t}, z_{i,t} \in \{0, 1\} \tag{2.9}$$

$$g_{i,t} >= 0, \tag{2.10}$$

where we have following denotations:

- $g_{i,t}$ is a continuous variable that represents the MWh of energy produced by generator i in period t,
- $y_{i,t}$ is a binary variable that is 1 if generator i is started at the beginning of period t, 0 otherwise,
- $z_{i,t}$ is a binary variable that is 1 if generator i is shut down at the beginning of period t, 0 otherwise,
- $u_{i,t}$ is a binary variable that is 1 if generator i is on during period t, 0 otherwise,
- $F_{i,t}$ is the fixed cost in \$/period of operating generator i in period t,

- $C_{i,t}$ is the variable cost function of generation for generator i in period t,
- $S_{i,t}$ is the cost of start up for generator i in period t in \$,
- Q_i is the upper bound in MWh for energy generated with generator i in every period,
- q_i is the lower bound in MWh for energy generated with generator i in every period,
- L_i is the minimum up time for generator i when it is started up,
- l_i is the minimum down time for generator i when it is shut down,
- d_t is the demand in period t,
- $1, ..., I$ is the set of generators,
- $1, ..., T$ is the set of time periods.

Then given a set $\{i = 1, ..., I\}$ of generators and a set of time periods $\{t = 1, ..., T\}$, the objective function (1.1) is to minimize the total operating costs, which is the sum of fixed operating cost $F_{i,t}$,the variable generating cost $C_{i,t}$, and the start up cost $S_{i,t}$, for all the generators in all time periods. In constraint (2.2) we have that the energy produced by all generators be greater or equal to the demand d_t in each period. In constraint (2.3) and (2.4) we have that the generators must operate within their maximum and minimum generation bounds. Constraint (2.5) and (2.6) controls when the generators are started up and shut down. And in constraints (2.7) and (2.8) we have that the generators must follow the minimum up time and down time requirements.

2.2 Dynamic Programming

The Dynamic Programming formulation of UC problems searches the solution space that consists of the units status for an optimal solution. The search can proceed in a forwrad or backward direction when the model is deterministic. However only the back direction can be used to solve the stochastic problems. Since in this thesis all the scenarios are considered deterministic, forward dynamic programming are used. The time periods of the study horizon are known as the stages of the DP problem, typically each stage represent one hour of operation. The combinations of

units that are on within a time period are known as the states of the DP formulation. Forward DP finds the most economical schedule by starting at the initial stage accumulating total costs, then backtracking from the combination of the least accumulated cost starting at the last stage and ending at the initial stage.

The mathematical formulation of the Forward Dynamic Program is given as follows:

Optimal Value Function $F(t, z)$ = Optimal total cost from period 1 to period t given that generators committed in period $t-1$ is given by the binary vector z.

Optimal Policy $A(t, z)$ = Optimal commitment for period t given that generators committed in period $t-1$ is given by y.

Recurrence Relation $F(t, z) = \min_i G(a, t) + F(t-1, a) + \max(a - z, 0)S_t$ where S_t is the startup cost vector.

Boundry Conditions $F(0, z) = 0$.

Answer $\min_a F(T, a)$

In this formulation, $G(a, t)$ stands for the optimal economic dispatching cost of commitment state indicated by a. The economic dispatching problem for each commitment state is given by

$$\text{Minimize:} \quad \sum_{i=1}^{I}\sum_{t=1}^{T} a_{i,t}F_{i,t} + \sum_{i=1}^{I}\sum_{t=1}^{T} C_{i,t}(g_{i,t}) \tag{2.11}$$

$$\text{subject to:} \quad \sum_{i=1}^{I} g_{i,t} >= d_t \tag{2.12}$$

$$g_{i,t} <= Q_i a_{i,t}, i = 1, ..., I \tag{2.13}$$

$$g_{i,t} >= q_i a_{i,t}, i = 1, ..., I \tag{2.14}$$

$$g_{i,t} >= 0, \tag{2.15}$$

DP builds and evaluates the complete decision tree to optimize the problem at hand. Thus, DP suffers from the "curse of dimensionality" because the problem grows rapidly with the number

Algorithm 2 Dynamic Programming for Unit Commitment

Step 0.
Initialization:
Load problem parameters.

Step 1.
Calculate generation cost of each state in each time period by solving the corresponding economic dispatching problem given in 2.11.
If the maximum generation capacity of certain state in certain time period is less than the demand or the minimum generation capacity of certain state in certain time period is greater than the demand, set generation cost as infinity so that the state will be rejected.
store generation cost matrix.

Step 2.
Calculate startup and shutdown cost for each state having 2^N paths from previous time period.

Step 3.
Calculate OVF for first state in first time period using recursive relation given in 2.2.Then check for min up and down time constraints, if violating set the transition cost to infinity so that path will be rejected. Calculate OVF for all states in all time periods.

Step 4.
Find the optimal path using Backward Induction.

of generating units to be committed.

2.3 Heuristic based on Priority List

Since conventional DP suffers from the "curse of dimensionality", it is computationally impractical when dealing with large systems. In order to significantly reduce computational complexity, only a small portion of the search space for the DP is considered.

This method arranges the generating units in a start-up heuristic ordering by operating cost combined with transition cost. The order of generators is chosen by evaluating the average generation cost for the generators when working at full capacity. The pre-determined order is then used to commit the units such that the system load is satisfied. Then the least costly units to operate are committed first and the most costly units to operate are committed last. In this case ,the problem is reduced by considering combinations of units sequentially turned on/off in priority list order [20].

When using the priority list to perform DP given N generators, only N states need to be considered in each decision period whereas the conventional DP needs to go over all 2^N possible

combinations of commitment states.

Algorithm 3 Heuristic based on Priority List

Step 0.
Initialization: Load problem parameters.

Step 1.
Generator Evaluation: Calculate the average generation cost when working at maximum capacity for all generators. Derive the priority list based on the average cost.

Step 2.
State Space Reduction: Following the priority list, reduce the total number of states by committing least costly generators first.

Step 3.
Calculate generation cost of each of the reduced state in each time period by solving the corresponding economic dispatching problem given in 2.11.
If the maximum generation capacity of certain state in certain time period is less than the demand or the minimum generation capacity of certain state in certain time period is greater than the demand, set generation cost as infinity so that the state will be rejected. store production cost matrix.

Step 4.
Calculate startup and shutdown cost for each reduced state having 2^N paths from previous time period.

Step 5.
Calculate OVF for first reduced state in first time period using recursive relation given in 2.2.Then check for min up and down time constraints. If violates, set the transition csot to infinity so that path will be rejected. Calculate OVF for all reduced states in all time periods.

Step 6.
Find the optimal path using Backward Induction.

2.4 Heuristics based on Predictive Modeling

When the Dynamic Programming approach is used to solve the UC problem, according to the formulation provided in 2.2, at each state in each stage, in order to calculate the optimal total cost, the economic dispatching sub-problem for each commitment state needs to be solved in each iteration. The economic dispatching problem given in 2.11 is an optimization problem which is computationally more expensive to solve compared to the other arithmetic operations in each iteration. This conjecture is confirmed in the computational experiments of 2.2. This provides motivation to reduce the computational effort by reducing the total number of optimization problems in the Dynamic Programming framework.

In this section, heuristics based on predictive modeling are provided which try to significantly

reduce the computational effort needed to solve the UC problem.

In the Dynamic Programming formulation, the most time consuming step is the one which calculates the economic dispatching cost of every commitment state in every state. The total number of economic dispatching sub-problems is $T * 2^N$ given that there are N generation units and T decision periods. Clearly this suffers from the "curse of dimensionality" because the computational complexity is exponential.

Instead of solving all the dispatching sub-problems, the heuristics first sample a polynomial number of commitment states in each period, solve the corresponding economic dispatching sub-problems, and the use the results of the sub-problems to formulate a predictive model, and then calculate the prediction of dispatching cost of the remaining commitment states using the derived predictive model. By doing this, the computational complexity is reduced to be polynomial, the dispatching cost matrix can be generated much more efficiently. In order for the heuristics to perform well, good predictive models are needed. In this section, several predictive modeling techniques,namely linear regression, principal components regression, and general regression neural networks, are employed.

Algorithm 4 Heuristic for Unit Commitment

Step 0.
Initialization:
Load problem parameters.

Step 1.
Generate a sample of a polynomial number of commitment states in each time period, calculate the generation cost of each state in the sample by solving the corresponding dispatching problem given in 2.11.

Step 2.
Using the results from Step 1, formulate a predictive model. Calculate the prediction of generation cost of the remaining commitment states. Store the generation cost matrix.

Step 3.
Calculate startup and shutdown cost for each state having 2^N paths from previous time period.

Step 4.
Calculate OVF for first state in first time period using recursive relation given in 2.2.Then check for min up and down time constraints. If violates, set the transition cost to infinity so that path will be rejected. Calculate OVF for all states in all time periods.

Step 5.
Find the optimal path using Backward Induction.

2.4.1 Heuristic Using Liner Regression

The most important component of our heuristics based on predictive modeling is to use partial generation cost data to predict the entire generation cost matrix. The predictors of the predictive model are the commitment variables and the response is the optimal generation cost derived from solving the economic dispatching model. The response of the model is a continuous numerical value, and linear regression is widely used in order to formulate this type of predictive model.

The objective of ordinary least square linear regression is to find the plane that minimizes the sum-of-squared errors (SSE) between the observed and predicted response:

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{2.16}$$

where y_i is the outcome and $\hat{y}_i$ is the model prediction of that sample's outcome. Mathematically, the optimal plane can be shown to be [21]

$$(X^T X)^{-1} X^T y, \tag{2.17}$$

where X is the matrix of predictors and y is the response vector. Equation 2.17 is also known as $\hat{\beta}$ in statistical texts and is a vector that contains the parameter estimates or coefficients for each predictor. This quantity 2.17 is easy to compute, so the linear regression technique is applied in the heuristic framework.

2.4.2 Heuristic Using Principal Components Regression

PCA is a commonly used data reduction technique. This method seeks to find the linear combination of the predictors, called principal components, which capture the most variance. The definition and derivation are illustrated in detail in 1.3. The primary advantage of PCA, and the reason that it has retained its popularity as a data reduction method, is that it creates components that are uncorrelated.

In this approach, when samples are generated, the principal components regression method

Algorithm 5 Heuristic for Unit Commitment using Linear Regression

Step 0.
Initialization:
Load problem parameters.

Step 1.
Generate a sample of a polynomial number of commitment states in each time period, calculate the generation cost of each state in the sample by solving the corresponding dispatching problem given in 2.11.

Step 2.
Using the results from Step 1, formulate a linear regression model and calculate the estimated coefficients for the predictors $\hat{\beta}$ using 2.17. Calculate the prediction of generation cost of the remaining commitment states using $\hat{y} = \hat{\beta}x$. Store the generation cost matrix.

Step 3.
Calculate startup and shutdown cost for each state having 2^N paths from previous time period.

Step 4.
Calculate OVF for first state in first time period using recursive relation given in 2.2.Then check for min up and down time constraints. If violates, set the transition cost to infinity so that path will be rejected. Calculate OVF for all states in all time periods.

Step 5.
Find the optimal path using Backward Induction.

is used to formulate the predictive model in order to reduce data dimension and reach better prediction precision.

2.4.3 Heuristic Using Artificial Neural Networks

Neural networks [22, 23] are powerful nonlinear regression techniques inspired by the theories about how the brain works. The outcome is modeled by an intermediary set of unobserved variables (called hidden variables). These hidden units are linear combinations of the original predictors. However these linear combination is typically transformed by nonlinear function such as the logistic function:

$$h_k(x) = g(\beta_{0k} + \sum_{i=1}^{P} x_j \beta_{jk}), \tag{2.18}$$

where

$$g(u) = \frac{1}{1 + e^{-u}} \tag{2.19}$$

The β coefficients are similar to regression coefficients; coefficient β_{jk} is the effect of the jth predictor on the kth hidden unit. A neural network usually involves multiple hidden units to

Algorithm 6 Heuristic for Unit Commitment using Principal Components Regression

Step 0.
Initialization:
Load problem parameters.

Step 1.
Generate a sample of a polynomial number of commitment states in each time period, calculate the generation cost of each state in the sample by solving the corresponding dispatching problem given in 2.11.

Step 2.
Using the results from Step 1, perform principal components analysis on the corresponding commitment state variables, and select the first principal components as the predictor matrix.

Step 3.
Using the results from Step 2, perform regression techniques on the predictor matrix derived from the principal components analysis. Calculate the prediction of generation cost of the remaining commitment states using $\hat{y} = \hat{\beta}x$. Store the generation cost matrix.

Step 4.
Calculate startup and shutdown cost for each state having 2^N paths from previous time period.

Step 5.
Calculate OVF for first state in first time period using recursive relation given in 2.2.Then check for min up and down time constraints. If violates, set the transition cost to infinity so that path will be rejected. Calculate OVF for all states in all time periods.

Step 6.
Find the optimal path using Backward Induction.

model the outcome. A more detailed discussion of neural network regression is given in 1.4.

The general regression neural networks technique is applied in the heuristic framework.

Algorithm 7 Heuristic for Unit Commitment using Neural Networks

Step 0.

Initialization:
Load problem parameters.

Step 1.

Generate a sample of a polynomial number of commitment states in each time period, calculate the generation cost of each state in the sample by solving the corresponding dispatching problem given in 2.11.

Step 2.

Using the results from Step 1, formulate a general regression neural networks model given in 1.4 and calculate the estimated coefficients. Calculate the prediction of generation cost of the remaining commitment states. Store the generation cost matrix.

Step 3.

Calculate startup and shutdown cost for each state having 2^N paths from previous time period.

Step 4.

Calculate OVF for first state in first time period using recursive relation given in 2.2.Then check for min up and down time constraints. If violates, set the transition cost to infinity so that path will be rejected. Calculate OVF for all states in all time periods.

Step 5.

Find the optimal path using Backward Induction.

Chapter 3

Computational Experiments and Numerical Results

In this chapter, all the formulation approaches discussed in the previous chapter will be implemented in order to solve the UC problem. First we would show the exact methods and their numerical results, then the heuristics will be implemented and tested in terms of accuracy and computational performance.

3.1 Problem Description

All the approaches were implemented using Matlab R2014a on a Windows 10 machine with Intel i5-4690 CPU @ 3.50GHz and 16GB memory.

The data of the test case used in this chapter is from [24]. The problem consist of 10 units, 24 periods. The demand pattern is shown in Figure 3.1 and Table 3.1.

Table 3.1. Load Pattern

hour	1	2	3	4	5	6	7	8	9	10	11	12
load	1025	1000	900	850	1025	1400	1970	2400	2850	3150	3300	3400
hour	13	14	15	16	17	18	19	20	21	22	23	24
load	3275	2950	2700	2550	2725	3200	3300	2900	2125	1650	1300	1150

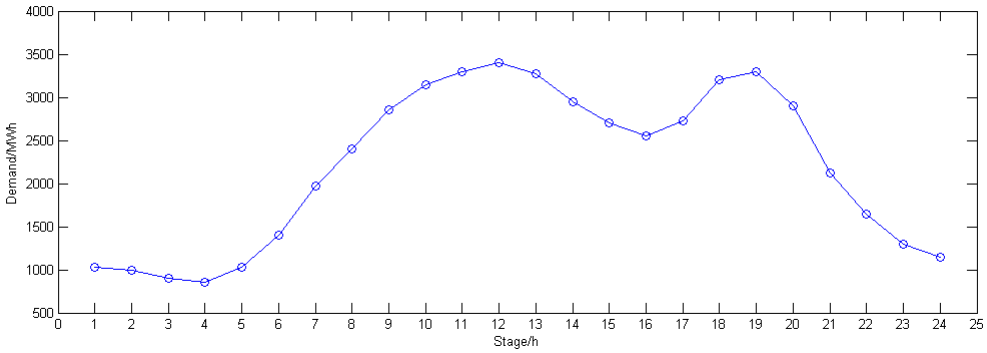


Figure 3.1. Demand Curve

.

We applied our solution approaches to the generating system as follows. The data on the generating units are provided in Table 3.2.

Table 3.2. Units Characteristics

Unit	Max	Min	Cost Function	Min Up	Min Down	Start-up Cost
1	1000	300	$0.00113x^2 + 9.023x + 820$	5	4	2875
2	400	130	$0.00160x^2 + 7.654x + 400$	3	2	2110
3	600	165	$0.00147x^2 + 8.752x + 600$	2	4	3050
4	420	130	$0.00150x^2 + 8.431x + 420$	1	3	2130
5	700	225	$0.00234x^2 + 9.223x + 540$	4	5	3000
6	200	50	$0.00515x^2 + 7.054x + 175$	2	2	2110
7	750	250	$0.00131x^2 + 9.121x + 600$	3	4	3250
8	375	110	$0.00171x^2 + 7.762x + 400$	1	3	1920
9	850	275	$0.00128x^2 + 8.162x + 725$	4	3	3150
10	250	75	$0.00452x^2 + 8.149x + 200$	2	1	1805

Table 3.3. Mixed Integer Programming Solution

Stage	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Unit 1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
Unit 2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Unit 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit 4	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Unit 5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit 6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Unit 7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit 8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Unit 9	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
Unit 10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0

3.2 Results of Mixed-Integer Programming

The Mixed-Integer Programming approach was implemented using CVX [25]. The model was solved using the MOSEK optimization solver [26]. After the problem was solved, an optimal policy was generated, with the commitment states for all time periods and the power to be generated by each unit.

The optimal states are shown in Table 3.3(For each generating unit, 1 is ON, and 0 is OFF).

The optimal generation cost for the 24-hour period:

$$V^*_{total} = 542167.23 \tag{3.1}$$

And the CPU time is 236.03 seconds.

3.3 Results of Dynamic Programming

The Dynamic Programming approach was implemented using Matlab, and the built-in *fmincon* optimization solver was used to solve the economic dispatching sub-problems. After the problem was solved, the optimal states were generated which are provided in Table 3.4.

The optimal generation cost for the 24-hour period:

$$V^*_{total} = 542167.17 \tag{3.2}$$

Table 3.4. Dynamic Programming Solution

Stage	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Unit 1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
Unit 2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Unit 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit 4	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Unit 5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit 6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Unit 7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit 8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Unit 9	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
Unit 10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0

Table 3.5. Heuristic based on Priority List Solution

Stage	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Unit 1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Unit 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit 3	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
Unit 4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit 5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Unit 6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit 7	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
Unit 8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit 9	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
Unit 10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

And the CPU time is 3668.30 seconds.

Since both the MIP and DP are exact solution approaches, their solutions are identical, however in our implementation experiments the MIP is much faster than the DP. According to the solution of the exact methods, Unit 3, Unit 5 and Unit 7 are never on during the 24-hour period, indicating that it is not efficient using these units.

3.4 Results of Heuristic based on Priority List

The heuristic based on priority list was implemented using Matlab, and the built-in *fmincon* optimization solver was used to solve the economic dispatching sub-problems. After the problem was solved, the optimal states were generated which are provided in Table 3.5.

The optimal generation cost for the 24-hour period:

$$V^*_{total} = 591042.30 \tag{3.3}$$

The error rate for this heuristic is:

$$error = 9.0\% \tag{3.4}$$

And the CPU time is 21.91 seconds.

The solution of this method indicates that Units 1,3,5,7,9 should be used in order to achieve lower cost. However according to the exact solution, Units 3,5,7 should not be on during the whole process, thus the solution of this heuristic has a relatively high error rate.

The heuristic based on priority list reduce the number of states in each stage from 2^N to be N, which is a solution method with linear computational complexity. Therefore the computation time for this heuristic is much less than the exact methods. On the other hand, since the economic dispatching problem was solved for every state this heuristic visits, the optimal total cost is the actual total cost when applying this solution to the generating system. As we can see the error rate is 9.0%, which is not great.

3.5 Results of Heuristic based on Linear Regression

The heuristic based on Linear Regression was implemented in Matlab, a states sample of size $2 \times N^2$ instead of 2^N was selected to calculate the economic dispatching sub-problem. Consequently the computational complexity was reduced to be quadratic. Since the heuristic consists of generating a random sample in each run, the solution results would be slightly different each time the heuristic was solved. The results provided in this section is only from one trial run. A solution is provided in Table 3.6.

After a solution was given, the actual generation cost for the 24-hour period is not known because the generation cost might be predictions instead of results from the economic dispatching sub-problems. Therefore in order to evaluate the solution, a simulation needs to be completed which takes the actual generation cost from previous methods as input to calculate the real cost to apply the solution.

Table 3.6. Heuristic based on Linear Regression

Stage	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Unit 1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
Unit 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit 3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
Unit 4	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
Unit 5	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
Unit 6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit 7	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
Unit 8	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
Unit 9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Unit 10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The optimal generation cost for the 24-hour period:

$$V^*_{total} = 584581.19 \tag{3.5}$$

The error rate for this heuristic is:

$$error = 7.8\% \tag{3.6}$$

And the CPU time is 1304.16 seconds.

The solution of this method indicates that Units 1,3,4,5,7,8,9 should be used in order to achieve low cost. However we know that using Units 3,5,7 is not efficient, so we have a significant error with this method.

From the results it can be concluded that the computational speed is faster than the exact DP approach, however worse than the heuristic based on priority list. On the other hand this approach provides a solution with better precision, since the expected total generation cost is lower.

Table 3.7. Heuristic based on Principal Components Regression

Stage	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Unit 1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Unit 2	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
Unit 3	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
Unit 4	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Unit 5	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
Unit 6	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0
Unit 7	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
Unit 8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
Unit 9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit 10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3.6 Results of Heuristic based on Principal Components Regression

The heuristic based on Principal Components Regression was implemented in Matlab, a states sample of size $2 \times N^2$ instead of N^2 was selected to calculate the economic dispatching sub-problem, and 5 principal components were selected to generate the prediction model. Consequently the computational complexity was reduced to be quadratic. Since the heuristic consists of generating a random sample in each run, the solution results would be slightly different each time the heuristic was solved. The results provided in this section is only from one trial run. A solution is provided in Table 3.7.

After a solution was given, the actual generation cost for the 24-hour period is not known because the generation cost might be predictions instead results of the economic dispatching sub-problems. Therefore in order to evaluate the solution, a simulation also needs to be completed which takes the actual generation cost from previous methods as input to calculate the real cost to apply the solution.

The optimal generation cost for the 24-hour period:

$$V^*_{total} = 583912.23 \tag{3.7}$$

The error rate for this heuristic is:

$$error = 7.1\% \tag{3.8}$$

Table 3.8. Heuristic based on Neural Networks

Stage	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Unit 1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
Unit 2	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
Unit 3	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
Unit 4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
Unit 5	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Unit 6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit 7	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
Unit 8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit 9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Unit 10	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

And the CPU time is 1305.37 seconds.

The solution of this method indicates that Units 1,2,3,4,5,6,7,8 should be used in order to achieve low cost. However we know that using Units 3,5,7 is not efficient, so we have a significant error with this method.

From the results it can be concluded that the computational speed is very close to the heuristic based on linear regression,faster than the exact DP approach, however worse than the heuristic based on priority list. In terms of performance of the solution, it is basically the same as the heuristic based on linear regression as well. It shows that in this particular case, the Principal Components Regression does not result in better solution than the Linear Regression model.

3.7 Results of Heuristic based on Neural Networks

The heuristic based on Neural Networks was implemented in Matlab, a states sample of size $2 \times N^2$ instead of N^2 was selected to calculate the economic dispatching sub-problem. Consequently the computational complexity was reduced to be quadratic. Since the heuristic consists of generating a random sample in each run, the solution results would be slightly different each time the heuristic was solved. The results provided in this section is only from one trial run. A solution is provided in Table 3.8.

After a solution was given, the actual generation cost for the 24-hour period is not known because the generation cost might be predictions instead results of the economic dispatching sub-problems. Therefore in order to evaluate the solution, a simulation also needs to be completed which takes the actual generation cost from previous methods as input to calculate the real cost

Table 3.9. Comparison of Approaches

Approach	MIP	DP	PL	LR	PCR	NN
Total Cost	542167	542167	591042	584581	583912	580653
Error Rate	0.0%	0.0%	9.0%	7.8%	7.7%	7.1%
Complexity	Exponential	Exponential	Linear	Quadratic	Quadratic	Quadratic
CPU Time	236.03	3668.30	21.91	1304.16	1305.37	1324.28

to apply the solution.

The optimal generation cost for the 24-hour period:

$$V^*_{total} = 580653.13 \tag{3.9}$$

The error rate for this heuristic is:

$$error = 7.1\% \tag{3.10}$$

And the CPU time is 1324.28 seconds.

The solution of this method indicates that Units 1,2,3,4,5,7,9,10 should be used in order to achieve low cost. However we know that using Units 3,5,7 is not efficient, so we have a significant error with this method.

From the results it can be concluded that the computational speed is very close to the other predictive modeling based heuristics,faster than the exact DP approach, however worse than the heuristic based on priority list. In terms of performance of the solution, it is basically the same as the heuristic based on linear regression and principal components regression as well.

3.8 Comparison of Approaches

In this chapter, 2 exact solution methods and 4 approximate heuristics were implemented in order to solve the same unit commitment problem. All of these approaches has their advantages and weaknesses. A comparison is provided in Table 3.9.

We can see that among the 2 exact approaches, the Mixed-Integer Programming is a lot more efficient in terms of computational time, and they both produce the same solution. One speculation is that the poor performance of DP is due to the low code efficiency of the Matlab loops, however the optimization of code is not the main concern of this thesis. The heuristic based on priority list is the fastest and provides a relatively good solution. The heuristics based on predictive modeling have similar computational time and solution performance in terms of error rate. They are better than the heuristic based on priority list, and slower too.

Chapter 4

Conclusion and Future Work

4.1 Conclusion of Thesis

The Unit Commitment problem is an important problem in the fields of power generation operations. Traditionally, Mixed-Integer Programming and Dynamic Programming are frequently used in order to solve the UC problem. However their computational complexity grows exponentially when the problem size grows. Therefore in this thesis, several heuristics with polynomial computational complexity were developed to solve the UC problem more efficiently. Additionally some preliminary computational experiments were conducted, all the approaches were implemented and solved, and their performance were evaluated.

The Forward Dynamic Programming method is used in this thesis because deterministic models are used. One would need to develop Backward Dynamic Programming models when applying DP to solve stochastic unit commitment problems.

Both the Mixed-Integer Programming method and the Forward Dynamic Programming method are developed in Chapter 2 and implemented and tested in Chapter 3. They both produce the same optimal solution as expected, and the computational performance is much better for the Mixed-Integer Programming method, mainly because it uses advanced commercial optimization solver while the Dynamic Programming method uses low efficiency Matlab code. The actual

difference between these two exact solution method cannot be concluded from our test.

The heuristic based on priority list performs well in the test which is very small in size compared to real-life cases. And its performance need to be further evaluated when applying to larger systems.

The heuristics based on predictive models performs relatively well in the computational experiments, however their actual performance also needs to be further evaluated. Among these heuristics, the heuristic based on linear regression and the one based on principal components regression have very little difference in terms of performance. In other words, PCA regression is not significantly better than linear regression. It is probably because PCA regression is mainly used to deal with correlation of dependent variables, there're not any significant correlations in our formulation. The heuristic based on neural networks seems to have better results, but it also need to be carefully evaluated and studied because of its complicated structure.

4.2 Future Work

Firstly all of the solution methods need to be further test in larger systems, in order to have better understanding of their performance.

Secondly the Dynamic Programming method can be improved greatly by implementing using a high level programming language with optimized code efficiency, so that its computational performance can be compared to Mixed-Integer Programming.

Thirdly alternative approaches to generate candidate states using priority list can be studied in order to improve its performance. When more candidate states are used in the model, it is likely that the error would be reduced.

In addition, all the models based on predictive modeling uses a one-time approximation in order to reduce complexity. An extension would be Approximate Dynamic Programming. An iterative algorithm can be formulated which updates the parameters of the predictive models in each step and achieve better approximation over the process.

Finally, all the method can be further studied in order to solve stochastic unit commitment problems.

Bibliography

[1] Goez, J., J. Luedtke, D. Rajan, and J. Kalagnanam (2008) "Stochasic Unit Committment Problem," .

[2] Padhy, N. P. (2004) "Unit commitment-a bibliographical survey," *Power Systems, IEEE Transactions on*, **19**(2), pp. 1196–1205.

[3] Garver, L. L. (1962) "Power generation scheduling by integer programming-development of theory," *Power Apparatus and Systems, Part III. Transactions of the American Institute of Electrical Engineers*, **81**(3), pp. 730–734.

[4] Muckstadt, J., R. C. Wilson, et al. (1968) "An application of mixed-integer programming duality to scheduling thermal generating systems," *Power Apparatus and Systems, IEEE Transactions on*, (12).

[5] Cohen, A. I. and M. Yoshimura (1983) "A branch-and-bound algorithm for unit commitment," *IEEE Trans. Power Appar. Syst.;(United States)*, **102**(2).

[6] Fisher, M. L. (1985) "An applications oriented guide to Lagrangian relaxation," *Interfaces*, **15**(2), pp. 10–21.

[7] Muckstadt, J. A. and S. A. Koenig (1977) "An application of Lagrangian relaxation to scheduling in power-generation systems," *Operations research*, **25**(3), pp. 387–403.

[8] Tseng, C.-L. (1996) *On power system generation unit commitment problems*, Ph.D. thesis, University of California, Berkeley.

[9] Tseng, C., C. Li, and S. Oren (2000) "Solving the unit commitment problem by a unit decommitment method," *Journal of Optimization Theory and Applications*, **105**(3), pp. 707–730.

[10] Powell, W. B. (2007) *Approximate Dynamic Programming: Solving the curses of dimensionality*, vol. 703, John Wiley & Sons.

[11] Howard, R. A. (1960) "DYNAMIC PROGRAMMING AND MARKOV PROCESSES.." .

[12] Zheng, Q. P., J. Wang, and A. L. Liu (2014) "Stochastic optimization for unit commitmentâĂŤA review," .

[13] Bertsekas, D. P. (1995) *Dynamic programming and optimal control*, vol. 1, Athena Scientific Belmont, MA.

[14] Jolliffe, I. (2002) *Principal component analysis*, Wiley Online Library.

[15] Specht, D. F. (1991) "A general regression neural network," *Neural Networks, IEEE Transactions on*, **2**(6), pp. 568–576.

[16] Jiang, B., Y. Zhang, S. Liang, X. Zhang, and Z. Xiao (2014) "Surface Daytime Net Radiation Estimation Using Artificial Neural Networks," *Remote Sensing*, **6**(11), pp. 11031–11050.

[17] Sheble, G. B. and G. N. Fahd (1994) "Unit commitment literature synopsis," *Power Systems, IEEE Transactions on*, **9**(1), pp. 128–135.

[18] Dreyfus, S. E. and A. M. Law (1977) *Art and Theory of Dynamic Programming*, Academic Press, Inc.

[19] Abhyankar, K., T. Fogarty, J. L. Kimber, A. Lin, S. Seo, and S. Takriti (1998) "Deterministic and stochastic models for the unit commitment problem," .

[20] Lowery, P. (1966) "Generating unit commitment by dynamic programming," *IEEE Transactions on Power Apparatus and Systems*, **5**(PAS-85), pp. 422–426.

[21] Kuhn, M. and K. Johnson (2013) *Applied predictive modeling*, Springer.

[22] Bishop, C. M. (1995) *Neural networks for pattern recognition*, Oxford university press.

[23] Ripley, B. D. (1996) *Pattern recognition and neural networks*, Cambridge university press.

[24] Shiina, T. and J. R. Birge (2004) "Stochastic unit commitment problem," *International Transactions in Operational Research*, **11**(1), pp. 19–32.

[25] Grant, M., S. Boyd, and Y. Ye (2008), "CVX: Matlab software for disciplined convex programming," .

[26] Mosek, A. (2010) "The MOSEK optimization software," *Online at http://www. mosek. com*, **54**.