UNIT COMMITMENT FOR COMBINED HEAT AND POWER PRODUCTION

Using Simulated Annealing

Þórhallur Ingi Halldórsson
s010261

Spring 2003
IMM DTU

Preface

This thesis is the final requirement for obtaining the degree: *Master of Science in Engineering.* The duration of this project was from the 1st of February to the 1st of July 2003. The work was carried out at the Operations Research section at Informatics and Mathematical Modeling, Technical University of Denmark. The supervisor of this project was René Victor Valqui Vidal.

I would like to thank my supervisor for the assistance throughout the project and Hans Ravn at Elkraft for assisting in the initial stages of the project.

Special thanks goes to Gauti Kjartan Gíslason for the cooperation during the project and Bergþór Jónson for computer related assistance. Finally I would like to thank my girlfriend for tolerating me during the last few weeks of the project.

I would like to point out that appendixes and source codes are not included with this report. They can however be obtained by contacting me by e-mail: *thih45@hotmail.com*.

Lyngby July 1st, 2003

Þórhallur Ingi Halldórsson
s010261

Abstract

In this project, Simulated Annealing is used to solve the unit commitment problem (UCP) for combined heat and power production system. The UCP involves scheduling the on/off status of units in a power production system in order to meet load demand while satisfying operational constraints. The UCP is a combinatorial optimization problem. A sub-problem of the UCP is the economic dispatch problem (EDP) which determines the production output for a given unit schedule. In this project the EDP is formulated as a linear programming problem.

A complicating factor in solving the EDP is that the power production system is composed of 4 different unit types were heat and power production is dependent and the number of constraints is large. The solution approach used, was to implement a fast heuristic method that solves the EDP. The heuristic was tested in comparison to an optimal solver and was found to produce near optimal solutions in all cases.

Simulated Annealing (SA) is used to solve the unit scheduling. A method that generates random feasible solutions for SA was implemented. SA was then tested on different sized problems and the results compared to optimal or lower bound solutions calculated in GAMS. Near optimal solutions were obtained with SA and the computation time was considerably faster for SA than GAMS. Finally, the performance of SA was compared to the performance of Tabu Search implemented by a fellow student working on the same project.

Keywords:

Unit commitment, economic dispatch, heuristic, combinatorial optimization, Simulated Annealing, combined heat and power, optimization.

List of abbreviations

- B&B: Branch and Bound
- EDP: Economic Dispatch Problem
- CHP: Combined Heat and Power
- DP: Dynamic Programming
- GAMS: The General Algebraic Modeling System
- GAs: Genetic Algorithms
- LR: Lagrangian Relaxation
- LP: Linear Programming
- MIP: Mixed Integer Programming
- SA: Simulated Annealing
- TS: Tabu Search
- UCP: Unit Commitment Problem

Contents

Chapter 1

Introduction

One of the main objectives when controlling large power production systems, is to make the best use of available resources. To do that considerable planning is needed. On an hourly basis, this involves determining the production output of individual units in the production system in order to meet the requirements of a given load demand. The objective is in most cases to minimize the overall production cost. The largest contribution to the production cost is the fuel consumption. Other factors also contribute such as operation and maintenance cost. In recent years the environmental factors have also been given much more weight and taxes on pollutant gases such as CO_2 have been added. When the production from a power production system is being planed, there are two complicating factors:

- The load demand is not constant during the day.
- Units can not produce continuously from zero to their maximum capacity. The minimum production is therefore bounded by some non-zero value.

This has the implication that units in a given power production system are not continuously active and to save cost, they must be turned on and off depending on the load demand at each time.

The problem of finding the optimal on/off scheduling for units in a power production system, is called the *unit commitment problem* (UCP). A sub-problem of the UCP is to find the optimal production output for a given combination of active units. This problem is called the *economic dispatch problem* (EDP). The UCP is a mixed integer combinatorial optimization problem. The term *combinatorial* is usually reserved for problems in which the decision variables are discrete [16]. In this cases the discrete variables are binary variables that represent the on/off status of units in the system.

For moderately sized production systems, exact methods[1] such as dynamic programming (DP) or branch-and-bound (B&B) can be used to solve the UCP. For larger systems

[1] these are solution methods which can be guaranteed to converge to the optimal solution.

exact methods fail because the size of the solution space increases exponentially with the number of time periods (hours) and units in the system. As a result, the computation time of exact methods becomes impractical. In these cases heuristic methods can be used to produce near optimal solutions in a reasonable computation time.

Even though the difficult part of the UCP is the combinatorial part (the unit scheduling), the nature of the underlying production system is also important. In conventional systems, heat and electricity are produced in separate units. Another type of units are the combined heat and power (CHP) units, which generate electricity and thermal energy in a single integrated system. The fuel efficiency in such units can be as much as 80-90%, while conventional units have fuel efficiency of about 40%. Because of higher fuel efficiency CHP units are both more economical and less pollutant than conventional units. The weight of CHP units in power production systems has increased considerably over the past years. In Denmark, approximately 70% of the total energy requirement is produced in CHP units. In other countries it is much less e.g. 30% in Finland and 10% in Germany.

Compared to conventional systems, the UCP is much more difficult to solve when CHP units are included. The reason is that in CHP units, the production of heat and electricity is strongly dependent on each other. As a result the EDP is two dimensional and the variables (heat and electricity) are strongly correlated. For conventional systems this production is not dependent and the UCP can be solved separately for heat and electricity (EDP is one dimensional). Quite a lot has been written about the UCP for conventional power production systems in the literature [1, 3, 4, 8, 9, 10, 11, 12, 13, 14, 21, 24, 26, 28]. Less has been written about the unit commitment problem were CHP units are included [5]. One reason for this is that the increase in use of CHP units is relatively recent.

1.1 Background

In 1999 a project called *Balmorel* was started[2]. The purpose of the *Balmorel* project is to support modeling and analyses of the energy sector in the Baltic Sea Region. The main emphasis of the project is on power production in both conventional and in CHP systems. The analyses cover all countries bordering the Baltic Sea[3]. One result of the project is the development of the *Balmorel* model. The model is directed towards answering various policy questions concerning power production in the region. The model includes production units from the whole region and the time scale of the simulation can range from days up to several years. The model takes into account production output

[2]detailed information can be found at *www.balmorel.com*.

[3]Denmark, Estonia, Finland, Germany (northern part), Latvia, Lithuania, Norway, Poland, Russia (western part and Kaliningrad) and Sweden.

from individual units and the distribution of production between regions and countries. Economical aspects such as instillation of new production capacity and fluctuations of prices for heat and electricity are also considered. Potential users of the model include research institutions, transmission system operators and energy companies.

As the model can be applied for large number of production units and over a long time horizon, there are various assumptions that have to be made. Looking only at the production, one of these assumptions is that units can produce electricity and heat continuously from zero and up to its maximum capacity. For large systems this assumption is necessary if the problem is to be solved by exact methods. When both the size of the system and time scale is reduced, more detailed information start to become relevant. That includes among other things the opening schedule of different units in the system over a relatively short production period, hence unit commitment must be taken into account.

Since the start of the *Balmorel* project, a large amount of data has been collected. This includes detailed information about production units and other data necessary to formulate the unit commitment problem. The motivation of this project is to use these information to formulate and solve the unit commitment problem for some moderately sized CHP system. The fact that the underlying data contains information on real production systems makes the solution process both more interesting and challenging.

1.2 Purpose

The aim of this project is to model and solve the unit commitment problem for a realistic CHP system using heuristic methods. The data used in the model formulation comes from the *Balmorel* project and the system of choice is the eastern part of Denmark[4].

The reason for using heuristics, is that the method implemented in this project should handle large production systems, producing near optimal solutions in reasonable computation time. The method of choice in this project is *Simulated Annealing* (SA). To achieve reasonable computation time, a fast heuristic method that solves the EDP had to be constructed.

The assumptions made in the model formulation are based on the quality of the *Balmorel* data which assumes linear production cost and unit constraints. The heuristic results are compared to optimal or lower bound solutions. The optimal solver is CPLEX with GAMS connected as a user interface.

[4]defined in the *Balmorel* project as those production units that are east of the Great Belt.

1.3 Outline

The remainder of this report is divided into 6 different chapters:

In **chapter 2**, the power production system under consideration is presented and the unit commitment problem is formulated. The first part of the chapter is used to prepare the ground for the model formulation. Some general concepts of combined heat and power production are discussed and the assumptions made in the formulation are stated. Properties of different unit types are also presented. The second part of the chapter presents a mathematical model for the problem and some basic characteristics of the model are stated.

The subject of **chapter 3** is the implementation and testing of the heuristic method that solves the economic dispatch problem. A possible alternative to the heuristic approach is given before describing the general principles that the heuristic is based on (section 3.2). Section 3.3 gives a more detailed technical description of the implementation, for those who are interested in specific details of the implementation. The chapter is concluded by presenting test results and performance measurements.

Chapter 4 gives an overview of some of the solution methods for the UCP that have been reported in the literature. The main purpose of this chapter is to give references and to present some alternatives to the solution method chosen. The 10 different methods discussed are divided into three main categories: *Exact-*, *relaxation-* and *heuristic methods*. The principles behind each method are described and the advantages and disadvantages of each method are stated. The chapter ends by giving the argument for the selection of SA. In connection to the contents of other chapters in this report, the discussion of the heuristic methods is the most relevant (section 4.3).

The subject of **chapter 5** is the theory and implementation of Simulated Annealing. The first part of the chapter discusses the theory and some basic concepts of the method. The remainder of the chapter is devoted to the implementation related decisions. This includes the selection of a cooling schedule, generation of neighborhood solutions and other implementation details. The argument for the selections made are given.

Experimental results are finally presented in **chapter 6**. The first part of the chapter focuses on the tuning and testing of Simulated Annealing. The performance of the method is then tested on the problem formulated in chapter 2. Special emphasis is put on investigating the influence of parameters that had to be estimated in the model formulation. The effect of time horizon and number of units is also reported. The chapter is concluded by giving a short comparison of the results obtained in this report and the results obtained by a fellow student working on the same project, using a different method.

Final conclusions on the results from this project are drawn in **chapter 7**.

The model formulation and the implementation of the heuristic presented in chapter 3 was a combined work made by me and Gauti Kjatan Gíslason (s011078). Chapters 2, 3, and *some parts* of chapter 4 are for that reason to a large extent similar. Conclusions drawn could be different from time to time.

Chapter 2

Case Study and Model Formulation

This chapter is divided into two main sections. The subject of the first section are the properties of the power production system. First, a short introduction on the general properties of combined heat and power production is given. Then the power production system is introduced. The assumptions made about the distribution network, demand and production cost are presented. A short description on the different unit types is then given and the constraints that production in each unit must satisfy are formulated. Based on that discussion, the second part of this chapter is devoted to the formulation of a mathematical model, which is used in later chapters to solve the unit commitment problem. Some properties of the model are also defined shortly. The main focus of the formulation is on:

- **Unit commitment**. This involves the binary on/off status of different units, start-up cost and constraints on unit opening and closing schedules.
- **Properties of production units**. This involves the constraints that production in each unit must satisfy.

All the data used in the model formulation originates from the *Balmorel* project. The region chosen in this formulation is the eastern part of Denmark. Within the region under consideration, the data does not contain information about exact position of individual units. When necessary information needed to formulate the unit commitment problem is missing in the data, a qualified guess is used. The purpose of the formulation is to represent a realistic CHP system for which the unit commitment problem is solved. Exact relation to the region chosen is not given much weight.

2.1 Case study

Combined heat and power production (also known as cogeneration) is a way to increase the efficiency of power plants. Standard electricity units effectively use around 40% of the fuel they burn to produce electricity. The rest of the energy ends up being wasted in the form of hot air. In CHP production this wasted heat is captured and used to produce hot water and steam. These products can then be used for district heating systems and for industrial applications. The fuel efficiency of CHP units can be as much as 80-90%. This makes CHP units obviously more economical than conventional units. Another advantage is the substantial reduction in the emission of pollutant gases such as carbon dioxide, sulphur dioxide and nitrogen oxides. The reduction is solely because CHP units use fuel more efficiently in producing heat and electricity than conventional heat boilers and electricity units combined. Figure 2.1 shows an energy-efficiency comparison between a CHP unit and a conventional electricity producing unit

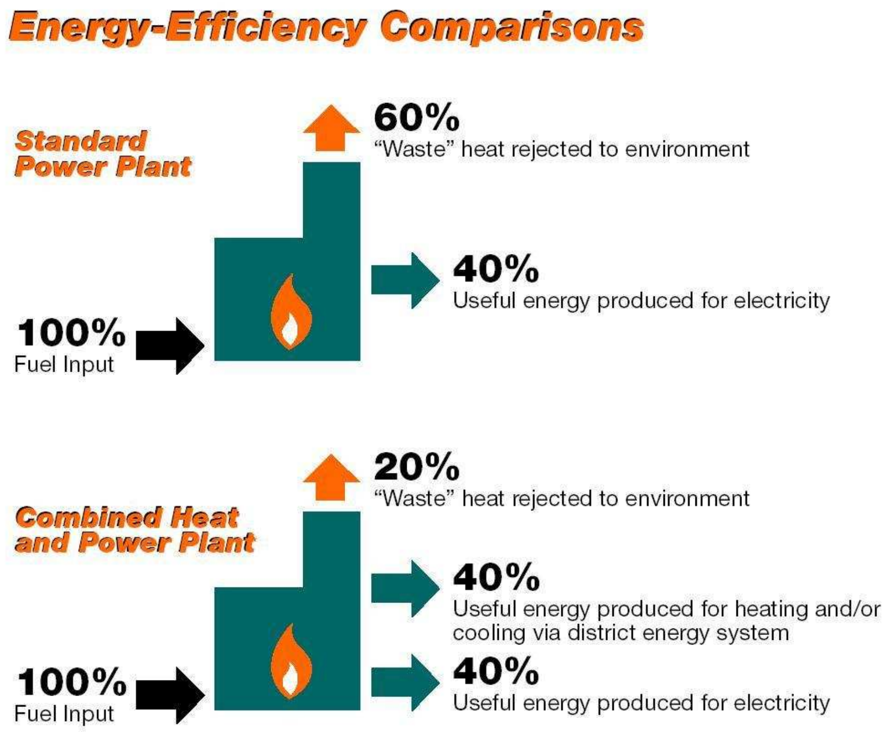

Figure 2.1: The figure shows the difference in efficiency between a conventional electricity producing unit and a CHP unit. Borrowed from the website of International District Energy Association, *www.districtenergy.org/chp.htm*.

Another difference between CHP units and conventional electricity units, is their positioning. Since electricity can be transported over long distances, the position of electricity units with respect to the consumer is not crucial. On the other hand, the distribution of heat is much more sensitive. To avoid substantial losses, heat is only distributed over short distances. Because of this loss in heat, CHP plants must be located close to the consumer. The electricity produced in CHP units can of course be transported over long distances if transmission network is available. Usually though, electricity is sold directly to the nearest consumer, which in some cases reduces the need for high capacity transmission network.

2.1.1 Properties of the power production system

The power production system in the eastern part of Denmark takes place in five different unit types which together make up to a total of 25 units[1]. Approximately 60% of the total electricity capacity is in CHP units. There are two types of CHP units in use and they are called *Back pressure-* and *Extraction units*. The difference between these two unit types lies in the way they utilize the excess heat from the electricity production. Units which produce electricity only are called *Condensing units* and units which only produce heat are called *Heat only boilers*. In addition to these four types of units, *windmills* are also used to produce electricity. For obvious reasons, windmills can not be controlled in the same way as normal power plants and are therefore not subjected to unit commitment. That means that these units are always considered active. More detailed description of individual unit types is given in section 2.1.2.

The distribution network

The units in the region under consideration belong to two different areas, urban and rural, based on their geographical position. In the urban area the number of units is 10 but 15 units are positioned in the rural area. The production capacity of the rural units is generally smaller than in the urban area. This is directly related to the difference in population density of the two areas. The heat produced in each area is distributed within that area. The electricity produced in both areas does however go to the same distribution network and is distributed over the whole region. The reason for splitting the heat production in the region, lies in the fact that heat can only be transported over short distances without significant losses.

In practice there is one important difference between the two areas. The heat producing units in the rural area, usually provide heat only to the population in their nearest neighborhood. This means that production load can not be distributed between these units based on economical reasons. The production in these units is controlled by the heat demand in their nearest neighborhood and since the heat demand is almost always non-zero these units are always active and not subjected to unit commitment.

The heat distribution network in the urban area is however assumed to be connected to all production units and capable of delivering heat to all consumers. Production load can therefore be assigned to the urban units based on economical reasons in each time period. Because of this difference in the heat distribution networks, units that should be subjected to unit commitment in this system are all units in the urban area and only the condensing units in the rural area, or a total of 14 units.

A problem composed of only 14 units is relatively small, and in order to be able to

[1] Based on the data from the Balmorel project

experiment more freely with problems of different difficulty, it was decided to ignore the restrictions of heat distribution in the rural area. The region under consideration is therefore assumed to be composed of one electricity distribution network with one electricity demand and two separate (non-connected) heat distribution networks, each with separate demand. As a result, unit commitment is imposed on all units or 25 in total. A schematic representation of the distribution network is shown in figure 2.2

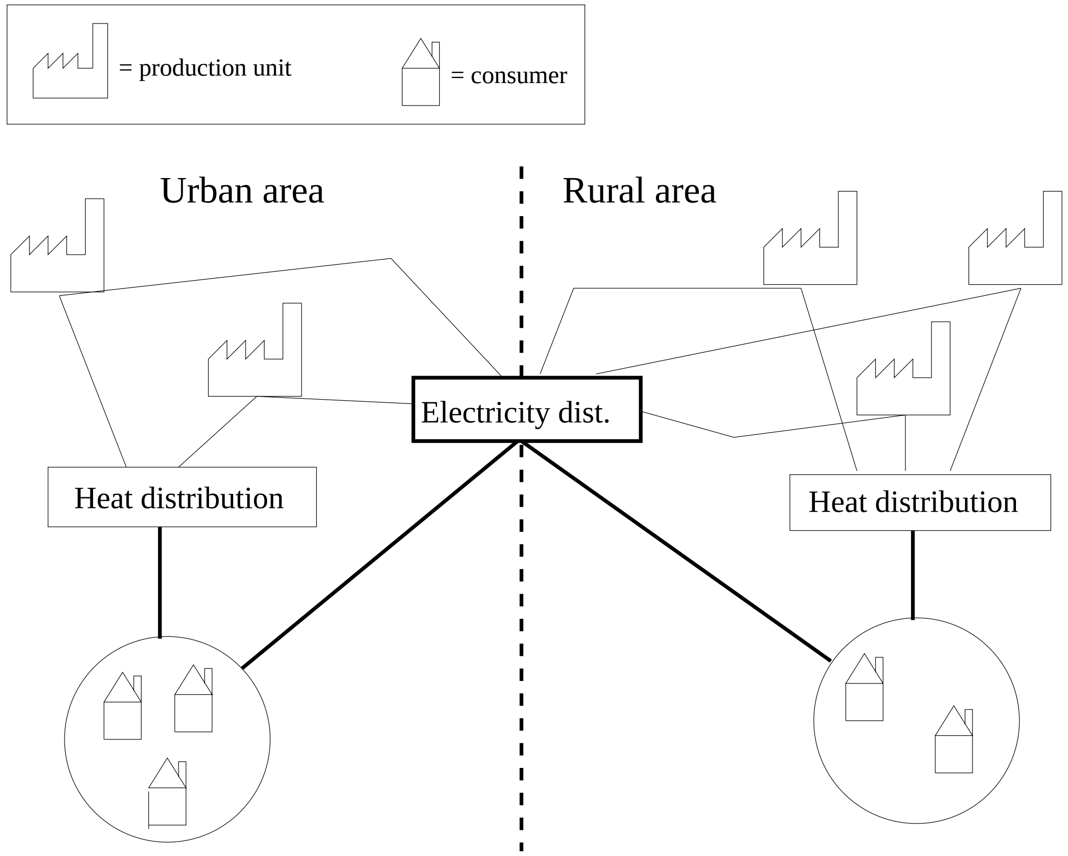

Figure 2.2: The figure shows the separate heat distribution network for the urban and rural areas. All units are connected to the same electricity distribution network.

Demand

In the model formulation, the demand for heat and electricity is deterministic and given on an hourly basis. The reason for choosing this time scale is that unit commitment is only relevant for small time intervals involving detailed production schedule over a short time horizon. Solving the UCP for more than 3-4 days is usually not practical both because units are subjected to malfunctions and the demand is difficult to predict accurately over long time periods.

In each time period the total production of heat and electricity must at least equal the demand in that time period. The demand constraint for electricity and heat are given in section 2.2, equations 2.24 and 2.25 respectively.

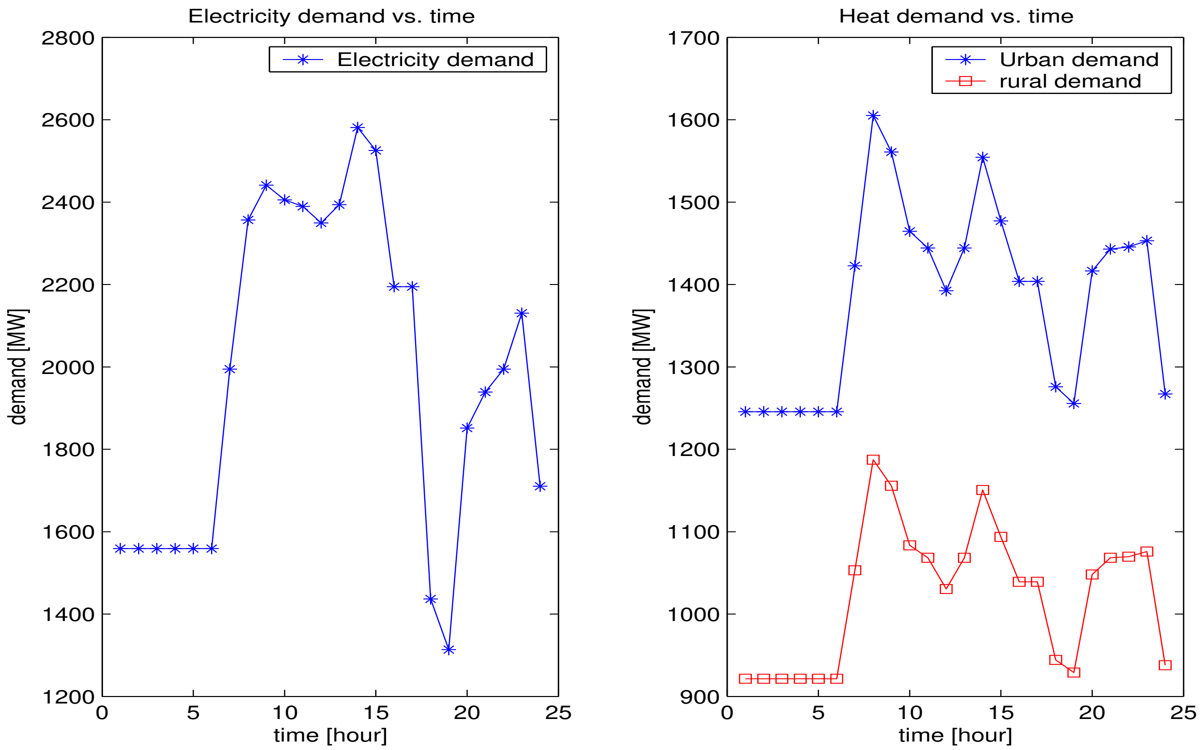

Figure 2.3: The figure shows the demand for electricity and heat (urban and rural area) for a typical winter day.

Another constraint related to the demand is the so called spinning reserve constraint. This constraint applies for both electricity and heat production and says that after the demand has been met, the total production capacity left in the system must at least be equal to a certain constant (the spinning reserve constant). The reason for this constraint is that if the demand suddenly increases, there must be extra capacity in the active units to take the unexpected load. The spinning reserve constraint for electricity and heat are given in section 2.2, equations 2.26 and 2.27 respectively.

Figure 2.3 shows the demand for heat and electricity on a typical winter day. This demand is used as a basis in the formulation. If longer time horizon than one day is used, the demand is given a random variation of 5% for heat and 10% for electricity between days. The fluctuation in heat demand is always smaller than for electricity both between and within days [2] (note the scale of the axis in figure 2.3).

[2]In contrast to electricity demand, heat demand changes dramatically between the seasons of the year.

Production cost

The production cost in each unit is composed of five different factors:

- *Fuel cost.* There are seven different types of fuel used: *Natural gas, coal, fuel oil, light oil, orimulsion, waste* and *straw* . In the model formulation, each unit uses one fuel type only.

- *Variable operation and maintenance cost.* This cost is calculated per megawatt produced and differs between units.

- *Start-up cost.* This cost differs significantly between units and is paid each time a unit is turned on.

- *Taxes on fuel consumption.* Calculation of this tax is based on the amount of fuel consumed for heat production only. In Denmark, the fuel consumption used for electricity production is not taxed directly. Consumer taxes on electricity are kept higher instead.

- *Taxes on emission.* The emission of carbon dioxide (CO_2), sulphur dioxide (SO_2) and various nitrogen oxides (NO_x) are subjected to taxes. This is done in order to reduce the amount released of these pollutants into the atmosphere. The amount of these gases released is highly dependent on the fuel type used.

The weight of these five production costs can differ significantly between units. Old units usually have low fuel efficiency and therefore high fuel cost. Other units burn only waste or recycled material which is cheap fuel. Some units also have a mechanism to reduce the amount of emission. The start-up cost is not given for individual units in the *Balmorel* data and its value was estimated by using a qualified guess based on unit type and capacity. Further information about different cost terms are given in appendix A.

2.1.2 Properties of different unit types

The subject of this section are the properties of the five different unit types that are used in the model formulation. The principles behind the production in each unit type is described shortly and their *working area* is both formulated and represented graphically. For each unit type, the working area gives the physical boundaries that the production must satisfy. In this formulation the working areas are defined by a set of linear equations. The following notation is used in the discussion that follows:

- t: index for time periods, $t = 1, \dots, T$

- i: index for units, $i = 1, \dots, N$

- $e(i, t)$: amount of electricity produced in unit i at time period t, [MW]

- $q(i,t)$: amount of heat produced in unit i at time period t, [MW]
- $u(i,t)$: binary variable, 1 if unit i is active in time period t else 0

Condensing unit

This unit produces electricity only. Water is pumped under high pressure into a boiler where it is heated up to 540^oC which gives a high pressure steam. The steam is then sent to a steam turbine to produce electricity which then enters the distribution network. From the turbine, the steam is sent to a condenser where it condenses with help of cooling water (water or sea water). The cooled water is then returned under pressure to the boiler and reheated. Figure 2.4 shows in principle how a condensing unit works.

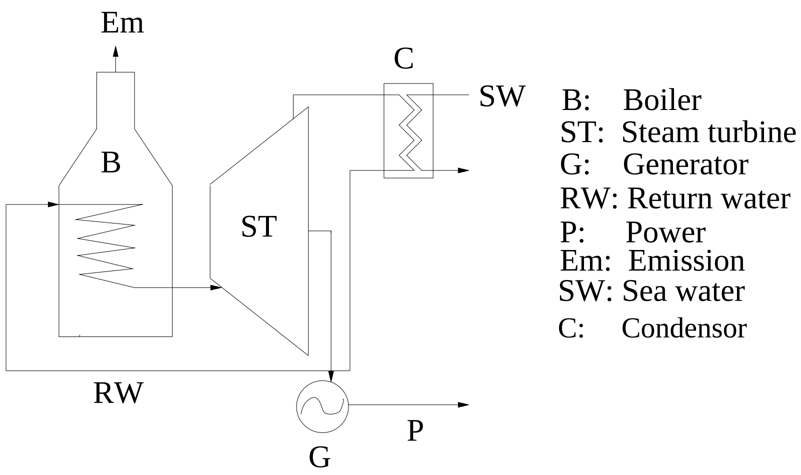

Figure 2.4: A schematic representation of a condensing unit.

The working area for condensing units is defined in equation 2.1 and shown graphically in figure 2.5. The working area for a condensing unit consists of the point $(0,0)$ and all points between the upper and lower bound on production (e_{min} and e_{max} respectively). When a unit is active, its production must at least be as great as e_{min} which is always greater than 0. The working point $(0,0)$ refers to the unit being closed. The working area for other unit types has this same property (see figures 2.7, 2.9, 2.11). The minimum production capacity is not given in the *Balmorel* model and its value had to be estimated for all unit types.

$$e_{min}u(i,t) \leq e(i,t) \leq e_{max}u(i,t), \qquad q(i,t) = 0 \tag{2.1}$$

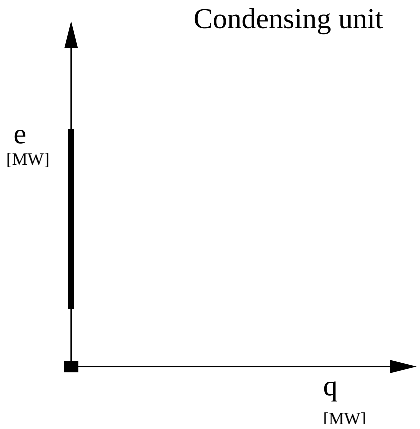

Figure 2.5: Working area for a condensing unit.

Heat only boiler

This unit generates heat only. Water is heated up in a boiler and then sent to a heat exchanger to heat up the district heating water. The district heating water is then distributed to both residential and industrial consumers. Figure 2.6 shows in principle how a heat only boiler works. In CHP systems, heat only boilers are mainly used to handle peaks in demand when it can not be satisfied with CHP units. The boilers can usually be started quite rapidly.

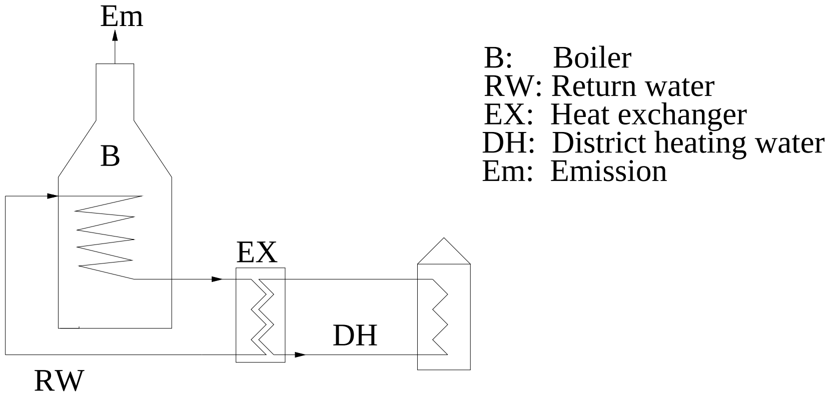

Figure 2.6: A schematic representation of a heat only boiler.

Similar to the condensing unit, the working area for the heat only boiler is defined by the upper and lower limits on production and the point $(0, 0)$. The working area is defined in equation 2.2 and shown in figure 2.7

$$q_{min}u(i,t) \leq q(i,t) \leq q_{max}u(i,t), \qquad e(i,t) = 0 \tag{2.2}$$

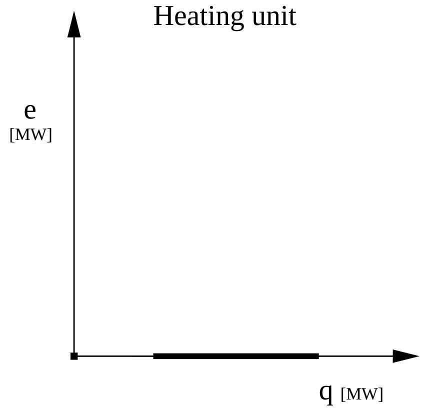

Figure 2.7: Working area for a heat only boiler.

Back pressure unit

This unit produces both heat and electricity. The unit works in a similar way as a condensing unit. One fundamental difference is that instead of sending the steam from the turbine to a condenser, as in the condensing units, the steam is sent to a heat exchanger where it is used to heat up district heating water. The final temperature of the district heating water after heat exchange is $100 - 200^{o}$C. Figure 2.8 shows in principle how a back pressure units works.

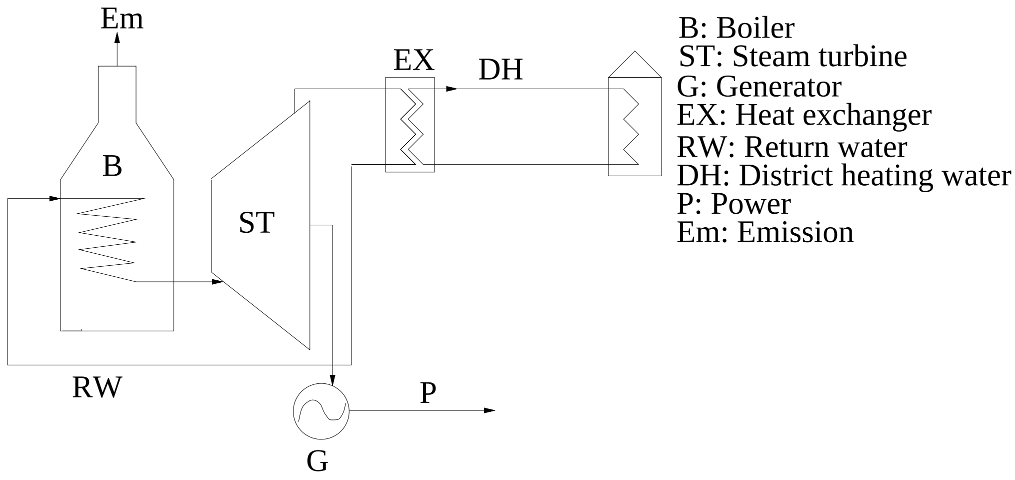

Figure 2.8: A schematic representation of a back pressure unit.

The working area for back pressure units is defined by equations 2.3-2.5 and is shown graphically in figure 2.9. The relationship between the amount of heat and electricity produced in a given working point $\big(e(i,t), q(i,t)\big)$ is given by the constant $c_m(i)$.

$$e(i,t) = c_m(i)q(i,t), \qquad 0 < c_m(i) < 1 \tag{2.3}$$

$$q_{min}u(i,t) \leq q(i,t) \leq q_{max}u(i,t) \tag{2.4}$$

$$e_{min}u(i,t) \leq e(i,t) \leq e_{max}u(i,t) \tag{2.5}$$

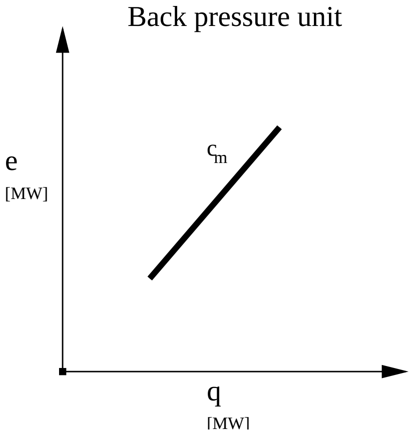

Figure 2.9: Working are for a back pressure unit.

Extraction unit

This unit generates both heat and electricity. It consists of a condensing unit and a back pressure unit. High pressure steam from the boiler is sent to the turbine to produce electricity. The steam from the turbine can then either be sent to a heat exchanger to heat up district heating water, or to a condenser where it is cooled down before it returns to the boiler again. Figure 2.10 shows in principle how an extraction unit works. Extraction units have relatively high production cost, but have the advantage that they can be started quite rapidly. In addition, the ratio between heat and electricity can be controlled instead of a fixed ratio as in back pressure units.

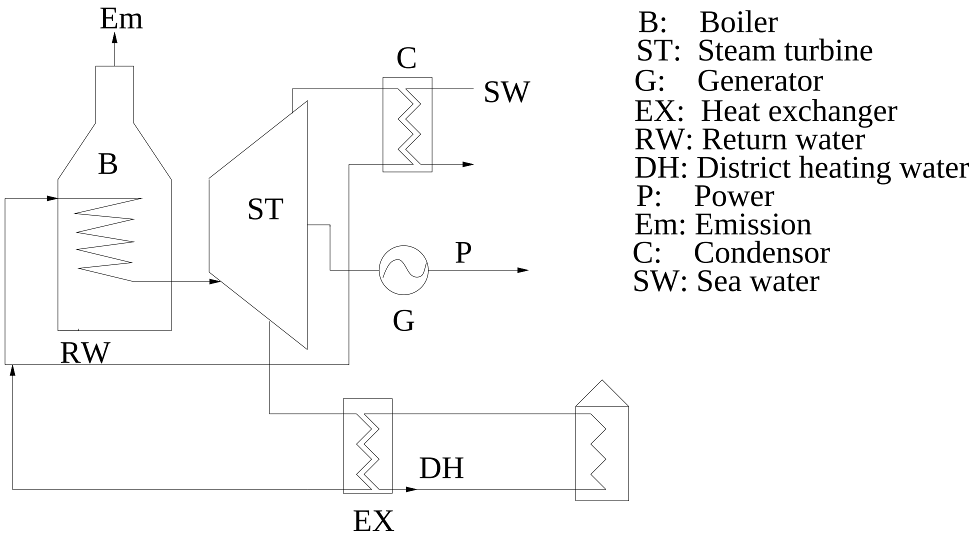

Figure 2.10: A schematic representation of an extraction unit.

The working area for extraction units is defined by equations 2.6-2.10 and is shown graphically in figure 2.11.

$$e(i,t) \geq \alpha(i)u(i,t) - c_v(i)q(i,t) \qquad 0 < c_v(i) < 1 \tag{2.6}$$

$$e(i,t) \leq \beta(i) - c_v(i)q(i,t) \qquad 0 < c_v(i) < 1 \tag{2.7}$$

$$e(i,t) \geq c_m(i)q(i,t) \qquad q(i,t) \geq 0, \quad c_m(i) < 0 \tag{2.8}$$

$$e_{min}u(i,t) \leq e(i,t) \leq e_{max}u(i,t) \tag{2.9}$$

$$q_{min}u(i,t) \leq q(i,t) \leq q_{max}u(i,t) \tag{2.10}$$

The constant $c_m(i)$ gives the ratio between heat and electricity production when all the steam from the turbine is sent to the heat exchanger. In that case the production takes place along the line given in equation 2.8. The iso-fuel constant $c_v(i)$, gives the ratio between electricity and heat for constant fuel consumption. The constants β and α are the intersection of the upper and lower iso-fuel line with the e-axis.

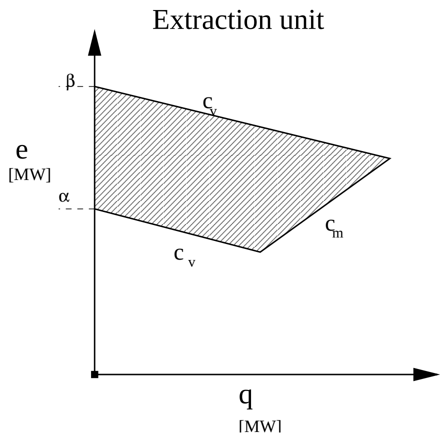

Figure 2.11: Working area for an extraction unit.

The feasible working area shown in figure 2.11 is defined by the point $(0,0)$ and all points on or below the line given in equation 2.7, points on or above the line given in equation 2.6 and all points on or left to the line given in equation 2.8

Windmills

The windmills used in this system produce electricity only and have maximum production capacity of 60 MW. The amount of electricity produced in each time period can not be controlled since it is totally dependent on the weather condition. The windmills are therefore neither subjected to unit commitment nor are they used when the economic dispatch problem is solved. Instead the average output of the windmills is predicted in each time period based on numerous measurements [3]. This estimated output is then subtracted from the electricity demand. The estimated output in each time period for the windmills is given with the problem data in appendix A.

Other unit constraints

In addition to the working area constraints, there are two other constraints that all units must satisfy. These constraints are called the *minimum up and down constraints.* The *minimum up constraint* states that a unit that has just been turned on must stay on for a certain number of time periods. This constraint arises from physical considerations associated with thermal stress on units. The *minimum down constraint*, on the other hand, states that a unit that has just been turned off can not be turned on for a certain number of time periods. The reason for this constraint is based on economic reasons intended to prevent excess maintenance and repair cost, due to frequent unit cycling [24].

The formulation of the minimum up and down constraint is given in equations 2.11 and 2.12. The following notation is used:

- $\tau^{on}(i,t)$: number of time periods since unit i was turned on (integer variable)
- $\tau^{off}(i,t)$: number of time periods since unit i was turned off (integer variable)
- $T^{on}(i)$: minimum *up* time for unit i
- $T^{off}(i)$: minimum *down* time for unit i

$$
\begin{aligned}
&\Big(u(i,t) - u(i,t-1)\Big)\Big(\tau^{on}(i,t-1) - T^{on}(i)\Big) \leq 0 \\
&\Big(u(i,t) - u(i,t-1)\Big)\Big(\tau^{off}(i,t-1) - T^{off}(i)\Big) \geq 0
\end{aligned}
\tag{2.11}
$$

where

$$
\begin{aligned}
\tau^{on}(i,t) &= \Big(1 + \tau^{on}(i,t-1)\Big)u(i,t) \\
\tau^{off}(i,t) &= \Big(1 + \tau^{off}(i,t-1)\Big)\Big(1 - u(i,t)\Big)
\end{aligned}
\tag{2.12}
$$

for $\quad t > 1$

The value of the minimum up and down parameters were not given in the *Balmorel* data. Instead a qualified guess was used. The minimum up and down times used in this formulation are between $2-5$ hours and they are assumed equal, $\tau^{on}(i,t) = \tau^{off}(i,t)$. The values of the parameters $\tau^{on}(i,t)$ and $\tau^{off}(i,t)$ are then assigned to different units in the following way:

[3] given in the *Balmorel* data

- Extraction units and heat only boilers are assigned 2 hours
- Condensing units with high production cost are also assigned 2 hours. These units are considered as peak units. Peak units are constructed in such a way that they have low start-up cost and can be started and closed rapidly at the expense of relatively high production cost.
- The values of $\tau^{on}(i,t)$ and $\tau^{off}(i,t)$ are assigned to other units based on their capacity. Large capacity means long minimum up and down times.

2.2 Model formulation

Last section gave a general description on how the power production system analyzed, is assumed to be functioning. Based on that discussion, the subject of this section is to formulate a mathematical model to solve the unit commitment problem. In addition to the assumption that the unit working areas can be defined by a set of linear equations, the objective function is also assumed to be linear with respect to heat and electricity production. All of the constraints used in the formulation have already been discussed and most of them have been formulated (except the demand and spinning reserve constraint). Before introducing the final model, the formulation of the production cost is discussed.

2.2.1 Formulation of production cost

As mentioned in section 2.1.1 the objective function is composed of five different cost terms. Each of these cost terms will now be formulated.

Fuel cost

The fuel cost calculations are based on the amount of fuel consumed in the production. The fuel consumption in unit i at time period t is given in equation 2.13.

$$\frac{e(i,t)+q(i,t)c_v(i)}{\gamma(i)} \tag{2.13}$$

Here $\gamma(i)$ is the fuel efficiency coefficient in unit i. The iso-fuel coefficient $c_v(i)$ is defined to be equal to 1 for all units except extraction units.

The nature of the fuel efficiency coefficient differ between units types. In condensing units $q(i,t) = 0$ and $\gamma(i)$ is given with respect to electricity production. For heat only boilers $e(i,t) = 0$ and $\gamma(i)$ is given with respect to heat production. In back pressure units $\gamma(i)$ is given with respect to both heat and electricity production. In extraction units $\gamma(i)$ is given with respect to electricity production only. The heat production is therefore converted into its equivalent electricity with the iso-fuel constant.

The fuel cost for unit i and time period t, $F_c(i,t)$, is then given by:

$$F_c(i,t) = f_c(i)\left(\frac{e(i,t)+q(i,t)c_v(i)}{\gamma(i)}\right) \tag{2.14}$$

$f_c(i)$ is the fuel cost coefficient for unit i and has the unit $[EUR/MW]$.

Variable operation and maintenance cost

This cost is paid per megawatt produced, for both heat and electricity. When this cost is calculated for extraction units, heat is converted into its equivalent electricity with the iso-fuel coefficient $(c_v(i))$. The operation and maintenance cost for unit i and time period t: $O_c(i,t)$, is given by:

$$O_c(i,t) = o_c(i)\Big(e(i,t) + q(i,t)c_v(i)\Big) \tag{2.15}$$

$o_c(i)$ is the operation and maintenance cost in unit i and has the unit $[EUR/MW]$.

Start-up cost

The start-up cost for unit i and time period t: $S_c(i,t)$, is paid each time a unit is turned on after the first time period and is given by:

$$S_c(i,t) = u(i,t)\Big(1 - u(i,t-1)\Big)s_c(i) \qquad t > 1 \tag{2.16}$$

$s_c(i)$ is the start-up cost of unit i and has the unit $[EUR]$.

The reason why the start-up cost is not paid in the first time period, is that it is assumed that the first time period belongs to a production pattern used previously.

Emission taxes

The emission tax in unit i and time period t: $E_c(i,t)$, is based on the amount of carbon dioxide, sulphur dioxide and various nitrogen oxides released into the atmosphere. The amount of these substances released is dependent on the amount of fuel consumed and fuel type. For unit i and time period t the tax is calculated by:

$$E_c(i,t) = \sum_{m \in XO}\Big[\quad t_E(m)\left(\frac{e(i,t) \quad + \quad q(i,t)c_v(i)}{\gamma(i)}\right)g_E(m,i) \quad\Big] \tag{2.17}$$

where

- m: is the emission type index
- $XO = \{CO_2, SO_2, NO_x\}$: is the set of emission types
- $t_E(m)$: is the tax coefficient for emission type m and has the unit $[EUR/Kg]$
- $g_E(m,i)$: gives the amount of emission type m released by unit i for each megawatt produced. It has the unit $[Kg/MW]$

Fuel taxes

Fuel tax in unit i and time period t: $T_c(i,t)$, is only paid in connection with heat production. The calculation of the tax is based on the amount of fuel consumed in the heat production:

$$T_c(i,t) = t_c\left(\frac{q(i,t)c_v(i)}{\gamma(i)}\right) \tag{2.18}$$

Here t_c is the fuel tax which is the same for all fuel types and has the unit $[EUR/MW]$. It is necessary to multiply with the iso-fuel coefficient ($c_v(i)$) since the fuel efficiency coefficient ($\gamma(i)$) is given with respect to electricity production in extraction units. Heat is therefore converted into its equivalent electricity production in the extraction units ($c_v(i) = 1$ for all other units).

The total production cost

The total production cost Z is obtained by summing up equations 2.13 - 2.18 for all time periods and units:

$$Z = \sum_{i=1}^{N}\left[\sum_{t=1}^{T}\left\{F_c(i,t) + O_c(i,t) + E_c(i,t) + T_c(i,t)\right\} + \sum_{t=2}^{T} S_c(i,t)\right] \tag{2.19}$$

Since this function is linear in $e(i,t)$ and $q(i,t)$ it can be simplified into three parts. The production cost for electricity, heat and start-up cost:

$$Z = \sum_{i=1}^{N}\left[\sum_{t=1}^{T}\left\{a(i)e(i,t) + b(i)q(i,t)\right\} + \sum_{t=2}^{T}\left\{u(i,t)\Big(1-u(i,t-1)\Big)s_c(i)\right\}\right] \tag{2.20}$$

Here $a(i)$ and b(i) are given by:

$$a(i) = \frac{f_c(i)}{\gamma(i)} + o_c(i) + \sum_{m\in XO}\left[\frac{t_E(m)g_E(m,i)}{\gamma(i)}\right] \tag{2.21}$$

$$b(i) = \frac{f_c(i)c_v(i)}{\gamma(i)} + o_c(i)c_v(i) + \frac{t_c(i)c_v(i)}{\gamma(i)} + \sum_{m\in XO}\left[\frac{t_E(m)g_E(m,i)c_v(i)}{\gamma(i)}\right] \tag{2.22}$$

2.2.2 The complete formulation

To conclude this chapter the final problem formulation is summarized in this section and the notation repeated:

Sets:

- $XO = \{CO_2, SO_2, NO_x\}$: is the set of emission types.
- $A \in \{U, R\}$: set of areas, U = Urban area, R = Rural area.

- $G(a)$: set of generation units in area a.

Indexes:

- t: time period, $t = 1, \ldots, T$.
- i: unit, $i = 1, \ldots, N$.
- m: emission type $m \in XO$.
- a: area, $a \in A$.

Variables:

- $e(i,t)$: electricity production in unit i at time period t, $[MW]$.
- $q(i,t)$: heat production in unit i at time period t, $[MW]$.
- $u(i,t)$: on/off status of unit i at time period t, (binary variable).
- $\tau^{on}(i,t)$: number of time periods since unit i was turned on (integer variable)
- $\tau^{off}(i,t)$: number of time periods since unit i was turned off (integer variable)

Parameters:

- $f_c(i)$: fuel cost in unit i, $[EUR/MW]$.
- t_c: tax on heat production, $[EUR/MW]$.
- $o_c(i)$: operating and maintenance cost in unit i, $[EUR/MW]$.
- $s_c(i)$: start-up cost in unit i, $[EUR]$.
- $g_E(m,i)$: emission type coefficient for emission type m in unit i, $[kg/MW]$.
- $t_E(m)$: tax on emission type m, $[EUR/kg]$.
- $D^e(t)$: electricity demand at time period t, $[MW]$.
- $D^q(t,a)$: heat demand in area a at time period t, $[MW]$.
- $R^e(t)$: spinning reserve for electricity at time period t, $[MW]$.
- $R^q(t,a)$: spinning reserve for heat in area a at time period t, $[MW]$.
- $e_{max}(i)$: maximum electricity production in unit i, $[MW]$.
- $e_{min}(i)$: minimum electricity production in unit i, $[MW]$.
- $q_{max}(i)$: maximum heat production in unit i, $[MW]$.
- $q_{min}(i)$: minimum heat production in unit i, $[MW]$.

- $T^{on}(i)$: minimum *up* time for unit i.
- $T^{off}(i)$: minimum *down* time for unit i.
- $\gamma(i)$: fuel efficiency coefficient for unit i.
- $c_v(i)$: iso-fuel coefficient for unit i.
- $c_m(i)$: relationship between electricity and heat production in unit i.
- $\beta(i)$, $\alpha(i)$: intersection of the upper and lower iso-fuel line with the e-axis.

The objective is then to minimize Z:

$$\min_{e,q,u} Z = \sum_{i=1}^{N} \Big[\sum_{t=1}^{T} \Big\{ a(i)e(i,t) + b(i)q(i,t) \Big\} + \sum_{t=2}^{T} \Big\{ u(i,t)\Big(1 - u(i,t-1)\Big) s_c(i) \Big\} \Big] \tag{2.23}$$

Subject to the following constraints:

Demand:
Electricity production must be greater or equal to the demand:

$$\textstyle\sum_{i=1}^{N} e(i,t) \geq D^e(t) \qquad \forall t \tag{2.24}$$

Heat production in each area must be greater or equal to the demand:

$$\textstyle\sum_{i \in G(a)} q(i,t) \geq D^q(t,a) \qquad \forall t, \quad \forall a \in A \tag{2.25}$$

Spinning reserve
There must at least be $R^e(t)$ megawatts available in the active units, once electricity demand has been fulfilled.

$$\textstyle\sum_{i \in G} e_{max}(i)u(i,t) \geq D^e(t) + R^e(t) \qquad \forall t \tag{2.26}$$

There must at least be $R^q(t,a)$ megawatts available in the active units in area a, once heat demand has been fulfilled.

$$\sum_{i \in G(a)} q_{max}(i)u(i,t) \geq D^q(t,a) + R^q(t,a) \qquad \forall t, \quad \forall a \in A \tag{2.27}$$

Minimum up and down constraints:
The minimum number of time periods a unit must be closed after being turned off, **or** open after being turned on is given by:

$$\Big(u(i,t) - u(i,t-1)\Big)\Big(\tau^{on}(i,t-1) - T^{on}(i)\Big) \leq 0 \qquad \forall i, \quad \forall t > 1 \tag{2.28}$$

$$\Big(u(i,t) - u(i,t-1)\Big)\Big(\tau^{off}(i,t-1) - T^{off}(i)\Big) \geq 0 \qquad \forall i, \quad \forall t > 1 \tag{2.29}$$

where

$$\tau^{on}(i,t) = \Big(1 + \tau^{on}(i,t-1)\Big)u(i,t) \tag{2.30}$$

$$\tau^{off}(i,t) = \Big(1 + \tau^{off}(i,t-1)\Big)\Big(1 - u(i,t)\Big) \tag{2.31}$$

Working area constraints:
For all active units, electricity and heat production must be between their upper and lower bounds.

$$e_{min}(i)u(i,t) \leq e(i,t) \leq e_{max}(i)u(i,t) \qquad \forall t, \quad \forall i \tag{2.32}$$

$$q_{min}(i)u(i,t) \leq q(i,t) \leq q_{max}(i)u(i,t) \qquad \forall t, \quad \forall i \tag{2.33}$$

For all active *back pressure units*, the following equality must in addition hold:

$$e(i,t) - c_m(i)q(i,t) = 0 \qquad \forall t \tag{2.34}$$

For all active *extraction units*, the following inequalities must hold

$$e(i,t) \geq a(i) - c_v(i)q(i,t) \qquad \forall t \tag{2.35}$$

$$e(i,t) \leq b(i) - c_v(i)q(i,t) \qquad \forall t \tag{2.36}$$

$$e(i,t) - c_m(i)q(i,t) \geq 0 \qquad \forall t \tag{2.37}$$

2.2.3 Characteristics of the model

A mathematical model for the problem has now been formulated. The rest of this report focuses on the implementation and development of a heuristic method that solves this problem and presents the results. Based on the formulation above, some of the characteristics of the model will now be addressed shortly. The purpose is to make the discussion of the problem in later chapters more clear.

The formulated model, involves minimizing the objective function Z with respect to the three decision variables e, q and u, while satisfying all constraints. The main objective with the model formulation is to include enough information so that the quality of different unit combinations can be estimated. The main weight of the optimization is therefore on the binary decision variable u. The solution with respect to the continuous variables e and q is used to estimate the quality of different unit combinations. The problem of minimizing the objective function Z with respect to these variables is called the economic dispatch problem and is a sub-problem of the unit commitment problem. Once a good unit combination is found, the economic dispatch problem can be re-optimized using a more detailed model. The objective of the unit commitment problem, is therefore primarily to find minimal turn-on and turn-off schedule for the power production system.

In this formulation the economic dispatch problem is defined as follows:

For a given unit schedule, minimize Z with respect to e and q **without** start-up cost, subject to

- The demand constraint (equations 2.24-2.25)
- The working area constraints (2.32-2.37)

The solution f is then used when the combinatorial part of the unit commitment problem is solved. The combinatorial part is defined as:

Find the unit combination that gives the lowest value of Z (f+start-up cost) subject to

- The spinning reserve constraint [4] (equations 2.26-2.27)
- The minimum up and down constraints (equations 2.32-2.37)

[4]if the spinning reserve constraint is fulfilled the demand constraint can also be fulfilled

2.3 Conclusion

This chapter has focused on the properties of the power production system and the formulation of a model to solve the unit commitment problem. The assumptions made about the system analyzed such as the distribution system, production cost calculations and unit working area were introduced. All the data used for the model formulation originates from the *Balmorel* project and is in most cases used directly. Because the parameters for production cost and working area are linear in the *Balmorel* model, that assumption was used here directly. In the formulation, problem parameters such as unit capacity, start-up cost and minimum up and down times had to be estimated. These parameters were estimated by using a qualified guess, which means that their values might be debatable. The effect of these parameters on the problem is therefore examined in more detail in chapter 6. Next chapter however, focuses on the economic dispatch problem and how it is solved.

Chapter 3

Solving the Economic Dispatch Problem

The subject of this chapter is the construction and implementation of a heuristic method that solves the economic dispatch problem (EDP). The performance of this method with respect to time and solution quality is a crucial part in the solution process of solving the unit commitment problem. For that reason, the first part of this chapter discusses a possible alternative to the heuristic approach and an argument for the selection made is given. General principles behind the construction of the heuristic method are then discussed before presenting a detailed description of the heuristic method. The last section of this chapter is then devoted to testing the heuristic and presenting the test results.

3.1 Possible solution strategies

The economic dispatch problem (EDP) was defined in section 2.2.3 and its objective is to minimize production cost while satisfying demand and working area constraints for a given combination of active units. The solutions of the EDP are used to guide the solution method that solves the combinatorial part[1], to find a combination of units with the least production cost. Figure 3.1 shows the connection between the combinatorial part and the EDP for the unit commitment problem.

If the unit commitment problem is solved by exact methods the EDP should also be solved by exact methods. If not, there is a risk that the method might not converge to the optimal solution. Since the EDP is formulated here as linear, the exact method of choice is linear programming (LP).

Many practical problem instances of the UCP can however not be solved by exact methods and heuristic methods have to be used. Compared to exact methods, the number

[1] also defined in section 2.2.3

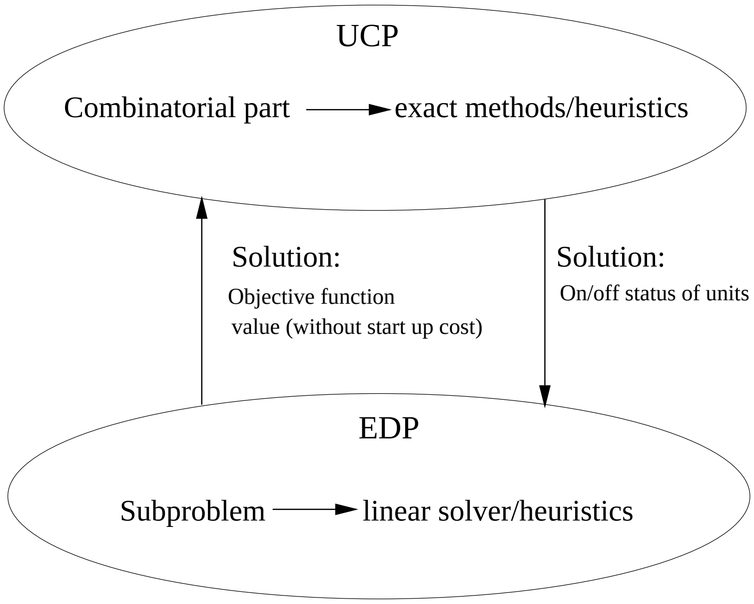

Figure 3.1: The figure shows the connection between the combinatorial part of the UCP and the EDP. When the combinatorial part is solved, solutions from the EDP are used to estimate the quality of different unit combinations.

of iterations a heuristic method needs to converge is quite high. An iteration number between 10.000-100.000 is for example not unusual. In those cases the use of LP can be an expensive function evaluation with respect to time. The reason being that for moderately sized systems, the EDP is relatively large linear programming problem which has hundreds of decision variables and constraints. Even though linear solvers work very fast, an execution time of 0.1 second[2], is not acceptable if the number of iterations needed is of magnitude 100.000. The effort used in solving the EDP with a linear solver can therefore lead to impractical computation times and there is a trade off between the quality of the EDP solution and the available computation time for the heuristic that solves the combinatorial part.

When heuristic methods are used to solve the UCP, the objective is to find a good combination of active units in a reasonable computation time. To achieve that an optimal solution to the EDP is not necessary and a good estimate is often sufficient. The use of heuristic methods to solve the EDP is therefore justifiable if the method produces near optimal solutions and outperforms an optimal solver with respect to computation time. The question that had to be answered in the initial stage of this project was therefore:

1. Is it feasible to use a linear solver to solve the EDP with respect to computation time?

2. Is it possible to use a fast heuristic method to solve the EDP?

[2]as an example

The feasibility of using a linear solver was first investigated. The solver of choice was CPLEX with GAMS connected to it as a user interface. The advantages of using GAMS connected to CPLEX is that it gives good flexibility for experimenting with problem data, compared to connecting CPLEX directly to the code of the UCP heuristic. The programming language used to implement the UCP heuristic is C++ in this case. The result of the experiment was that the GAMS → C++ connection was too slow. A single iteration took approximately 2 seconds which is far too much time if the number of iterations is in the range of few thousands. A large portion of the computation time is due to the overhead when GAMS is compiling and feeding data to CPLEX. This overhead could be reduced by connecting CPLEX directly to C++. Implementing such a connection is from a programming point of view quite complex and gives less flexibility in experimenting with problem data. It was also estimated that the computation time with CPLEX connected to C++ would still be around 0.1-0.4 seconds. It was therefore decided not to use a linear solver to solve the EDP and concentrate on implementing a fast heuristic instead. The other two reasons for this choice were:

- Very little has been written about the use of fast heuristics in connection with this type of problem, especially for CHP systems. It would therefore be interesting to see how a heuristic method could perform for this type of problem.
- If correctly implemented, the heuristic method could be extended to work on non-linear convex objective functions, as a long term objective. The implementation of a heuristic for the linear formulation of the EDP, could be used to estimate the feasibility of that approach.

The rest of this chapter discusses the implementation and testing of that heuristic. A description of the GAMS → C++ connection is given in appendix C.1

3.2 General principles of the EDP heuristic

There were 3 main objectives in the construction of the EDP heuristic:

- The method must converge fast in few iterations
- The method must be extendible for convex non-linear objective functions
- The quality of the final solution must be sufficient to distinguish between the solution quality of different combinations of active units

The first and the second point are quite easy to control. It is always possible to make a fast method by compromising the solution quality and making crude approximations. The extendibility to non-linearity is also a matter of construction. Implementing a solution method based on similar principles as *simplex* [6] would obviously not be extendable to non-linear cases. The third point however is the most difficult to control. Deviation from the optimal value are inevitable but it is important that the heuristic method estimates the change between two solutions[3] in the same way as an optimal solver. That is, given two solutions x_{k+1} and x_k it is important that if $f^*(x_k) > f^*(x_{k+1})$ then $f(x_k) > f(x_{k+1})$ where f^* denotes the optimal value and f the objective value obtained by the heuristic method. For a heuristic method, this property can usually not be derived mathematically and must be observed by testing.

In order to achieve these objectives, it was decided to use direct search method using greedy approach. Direct search methods have been used to solve the economic dispatch problem before [9] and the results with respect to computation time have been quite good. The nature of the power production system was however quite different, with only electricity production considered (1 dimensional system), non-linear objective function and unit commitment was not involved. What complicates the optimization in this case is that this system is 2 dimensional with four different types of units involved where some produce only heat (q) or electricity (e) and others produce both. The way the variables e and q can be changed is also dependent on the unit type. For condensing units and heat only boilers e and q, can be modified freely between their upper and lower bounds respectively. In back pressure units a change in e gives a fixed change in q and vice versa and in extraction units e and q can be modified independently inside the working area.

The problem that then arises, is how to compare the production cost in a non-CHP unit to a production cost of the same product in a CHP unit. It is for example not possible to compare cost coefficients for electricity production $a(i)$ in a condensing unit and CHP unit directly. In CHP units the cost coefficients for electricity and heat are dependent and can not be used to evaluate production cost for one product only. One possibility would be to assume that heat or electricity could be sold at a fixed price and subtract that cost from the total production cost in the CHP units. That would make the price coefficients independent. The problem is that the prices of heat and electricity fluctuate within a day and are difficult to estimate. This approach would therefore complicate the model. An easier way to compare the production costs between units is to use the following ratio:[4]

$$\frac{a(i)\Delta e_i + b(i)\Delta q_i}{v\Delta e_i + w\Delta q_i} \tag{3.1}$$

where

- $a(i)$ and $b(i)$ are the cost coefficients for heat and electricity from equation 2.23
- Δe_i is the increase in electricity production in unit i

[3] unit combinations

[4] For heat only boilers $\Delta e_i = 0$ and for condensing units $\Delta q_i = 0$.

- Δq_i is the increase in heat production in unit i
- v is a binary variable 0 if electricity demand is fulfilled, else 1
- w is a binary variable 0 if heat demand is fulfilled, else 1

This ratio is called the *gain* factor of unit i but is in fact the derivative of the cost function. The cost of producing either heat, electricity or both is divided by the resulting output (in MW). This ratio of production cost versus output gives the possibility of comparing the cost of different units, based on their production output. The reason for the binary variables v and w is that when either heat or electricity demand has been fulfilled, the contribution of that output is no longer taken into account. If the *gain* equals infinity, the production in that unit is no longer feasible.

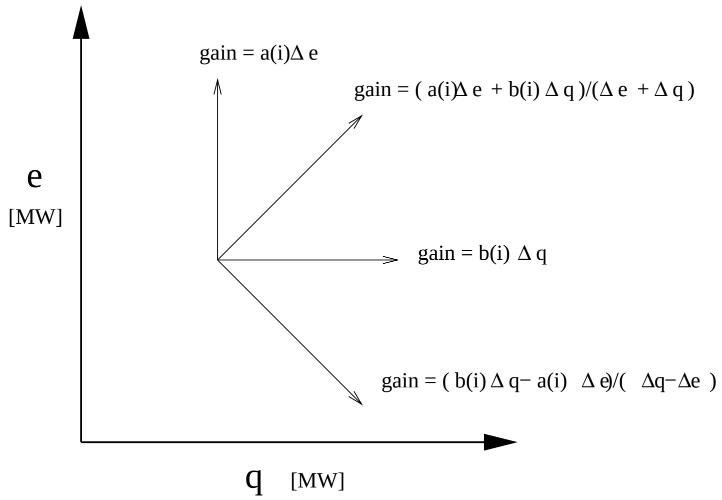

Figure 3.2: The figure shows how the value of the *gain* changes for different search directions.

In CHP units where e and q can be changed independently, the value of the *gain* is dependent on the search direction[5]. This is shown in figure 3.2. The question then arises how a search direction should be chosen. One possibility is to allow all possible search directions inside the unit working area. This would allow the production to be increased and decreased in the most suitable way during iteration. The other possibility is to allow only a limited number of search directions that are considered most feasible. In this implementation it was decided to limit the number of search directions. The two main reasons for this choice were:

- since the EDP is linear, the best search direction is always known (see section 3.3)
- By limiting the search possibilities the method becomes easier to implement and computationally more simple (faster)

[5]each search direction is defined by the values of Δe_i and Δq_i

The search strategy used in this implementation is to iterate in a forward direction from an initial solution until all demand constraints have been fulfilled. Decrease in production is in most cases forbidden. If a decrease in production would be allowed the whole process would be much more complicated. Decrease of one product would have to be compensated by an increase of the same product in another unit which requires further calculation and more iterations.

Finally it is worth mentioning that since the EDP is linear, the *gain* is constant for a given search direction. That means that when a unit is selected based on its *gain*, the production should be increased as much as possible. The EDP heuristic then works in the following way:

Initially, the method makes a list which holds the best possible *gain* in each unit. The initial production in each unit is its minimum capacity. In each iteration the method chooses the unit with the lowest *gain*[6] and increases the production in that unit as much as possible. The best *gain* for that unit is then recalculated and put back into the list. If in this iteration either heat or electricity demand has been fulfilled, the relevant binary variable v or w is set equal to 0 when the *gain* is recalculated. If the unit has reached its maximum capacity the unit is excluded from selection. This process is repeated until heat and electricity demand has been fulfilled.

It can happen that the final solution has too much production of heat or electricity. If that is the case the final solution is modified by decreasing the excess production (if it is possible) in the most expensive units. This modification of the final solution is highly dependent on the types of units in the system and is discussed in more detail in the next section. A pseudocode for the EDP method is as follows:

Begin
    Set $e(i) = e(i)_{min}$ *and* $q(i) = q(i)_{min}$ $\forall i$;
    Initialize(L);
    Repeat
        gain := Best(L);
        Increase $e(i)$, $q(i)$ *along the search direction until demand*
         is fulfilled or working area constraint is reached;
        Update(L);
    Until all demand is fulfilled;
     if excess production, then decrease that production if possible;
End;

[6] remember that the lowest *gain* gives the most output of e or q or both with respect to production cost.

Here L is the list holding the best *gain* for each unit.

The fact that the method iterates from minimum production until demand has been fulfilled makes it work quite fast. By iterating in this way it is not necessary to find an initial solution and then try to improve that solution. The method simply makes a good feasible solution in one go. The *gain* list (L) enables the method to distinguish between the cost efficiency of different units by choosing the unit with the best available direction of improvement in each iteration.

3.3 Description of the EDP heuristic

We are now in position to discuss how the method is implemented for the four different unit types that are used in the model formulation. As mentioned before, the unit with the best allowed *gain* is selected in each iteration and the production updated. For all units except extraction units, there is only one search direction possible . In extraction units the number of possible search directions are infinite. For this problem there are two factors that simplify the search for the best direction.

- The *gain* constant for each search direction.
- In the extraction units, the cost coefficient for heat production $b(i)$, is always smaller than the coefficient for electricity production $a(i)$.

This means that the best possible search direction in extraction units is the direction that gives as much heat production as possible and as small electricity production as possible. If the cost coefficient for electricity would be smaller than the heat coefficient, a direction with as small heat and large electricity production would give the best direction. If both coefficients are equal the search direction can be chosen arbitrarily. The best possible direction is therefore always along the boundary of the working area. This simplifies the choice of search direction significantly and in this implementation the number of search directions needed is only 4. To distinguish between different search directions a new parameter called *move* is defined. Figure 3.3 shows all allowed the *moves* for each unit type and the corresponding *gain*-values are shown in table 3.1.

The *gain*-values shown in table 3.1 are found using equation 3.1. It should be noted that when the *gain* is calculated, the step size for heat Δq_i is set equal to 1 and relevant electricity step Δe_i is calculated based on that value. For units that have only one product the step size is set equal to 1.

The starting point of the EDP method in each unit is marked with a dot in figure 3.3. For all units except extraction units the starting point is at the minimal production. For extraction units the starting point is at the intersection of the lower iso-fuel line with the e-axis. The selection of *move* is obviously trivial for all units except extraction

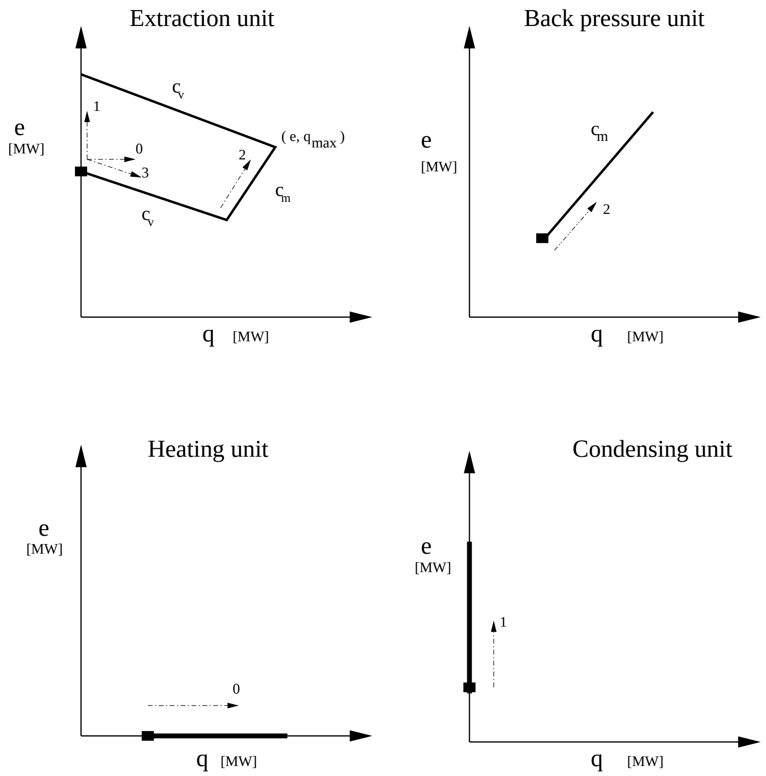

Figure 3.3: Allowed *moves* for each unit type.

units. Initially $moves = \{0,1,3\}$ are allowed for extraction units and $move = 2$ is only allowed once the c_m-line is reached. As mentioned before it is always most profitable to move along the edges of the working area. The moves along the lower iso-fuel line and along the c_m-line are therefore allowed. The move along the upper iso-fuel line is however forbidden. There are two reasons why the method does not allow this *move*:

- It complicates the computations significantly
- It complicates the unit commitment problem with respect to estimating capacity in active units.

These two reasons are actually interconnected. It is necessary to be able to estimate the total capacity of heat and electricity for all active units. If this is not possible, there is no way of deciding if a given combination of units is feasible with respect to demand and spinning reserve, without sending the solution to the EDP method and find out if it converges. That is computationally expensive.

In extraction units the maximum heat and electricity production takes place in two separate working points. It is obviously not possible to be in these two points simultaneously and a point of reference must be chosen to estimate the heat and electricity capacity. This point is chosen as $(e(i,t), q_{max})$. During iteration, this point must be reach-able for each unit that is open for selection. If this condition is not fulfilled there

move	Extraction	Back pressure	Condensing	Heat only
0	$\frac{b(i)}{w}$	Inf	Inf	$\frac{b(i)}{w}$
1	$\frac{a(i)}{v}$	Inf	$\frac{a(i)}{v}$	Inf
2	$\frac{a(i)+b(i)}{c_m v+w}$	$\frac{a(i)+b(i)}{c_m v+w}$	Inf	Inf
3	$\frac{-a(i)c_v+b(i)}{1v-c_v w}$	Inf	Inf	Inf

Table 3.1: The *gain* corresponding to each allowed move for the 4 different unit types. If a given search direction is not available for selection its gain is set equal to infinity (Inf).

is the risk that demand can not be fulfilled based on the initial capacity calculations. By not allowing a *move* along the upper iso-fuel line, this condition holds for the extraction units. If a *move* along the upper iso-fuel line was allowed, it would be necessary for the method to check the feasibility of lowering the heat production by considering the production status and available capacity in other units. This would make the iteration process much more complicated.

In each iteration the EDP method selects the unit with the best possible *move*. If that *move* belongs to an extraction unit the production is updated and new best *move* selected. The *move* and the corresponding *gain* is then added to the list that the method selects from. In **extraction units** the updating is done as follows.

- If $move = 0$ then increase q until boundary is reached or demand is met. If heat demand is met then electricity increase is only allowed ($move = 1$). Otherwise increase along the c_m-line is allowed ($move = 2$).

- If $move = 1$, then increase e until boundary is reached or demand is met. Exclude unit from further selection. This is done because the heat coefficient is always lower than the electricity coefficient and $move = 1$ is only selected when no more heat is needed.

- If $move = 2$ is selected, then production is increased along the c_m-line until demand is met or boundary is reached. If boundary is reached then exclude unit from further selection. In this case the method is stuck in the point (e, q_{max}). If electricity demand is met, stop and set $v = 0$. Recalculate the *gain*-value. If heat demand is met then $move = 1$.

- If *move* = 3, increase q and decrease e along the lower iso-fuel line until boundary is reached or heat demand is met. If heat demand is met, set *move = 1*, else set *move = 2*.

A pseudocode for the updating process in the extraction units then is as follows[7]:

Extraction unit

```
if( best move == 0 )
    Increase q until boundary is reached or demand is met.
    if( boundary is reached AND heat demand is not met )
        move = 2
    else
        move = 1
if( best move == 1 )
    Increase e until boundary is reached or demand is met.
    gain = Inf
if( best move == 2 )
    Increase e and q until boundary is reached or demand is met.
    if( boundary is reached )
        gain = Inf
    elseif( electricity demand is met )
        move = 2, where v = 0
    else
        move = 1
if( best move == 3 )
    Increase q and decrease e along the lower iso-fuel line
    until boundary is reached or heat demand is met
    if( boundary is reached AND heat demand is not met )
        move = 2
    else
        move = 1
```

The updating process for other units is much simpler than for extraction units. Production is at most updated twice in back pressure units but never more than once in condensing and heat only boilers.

In the **back pressure** units the production is increased until boundary is reached or demand is met. If boundary is reached then the *gain* is set as Inf which means the unit is no longer available for selection. If demand is reached for one product then either v or w are set to 0, depending on which demand is met. This is done to eliminate the relevant

[7] == means equal, = means assignment

output from the *gain* calculation. The unit is however still available for selection. The pseudocode for back pressure units is then given by

Back pressure unit

Increase ***e*** *and* ***q*** *until boundary is reached or demand is met.*
if(boundary is reached)
    $gain = Inf$
else
    $gain = \frac{a(i)c_m + b(i))}{c_m * v + 1 * w}$ *where v or w = 0*

For **condensing units** and **heat only boilers** the production is increased as much as possible. The gain is then always set to Inf so the unit is no longer available for selection. The reason why the unit is no longer available for selection after iteration is that either demand is fulfilled or boundary has been reached. The pseudocode for heat only boilers and condensing units is then given by:

Condensing unit

Increase ***e*** *until boundary is reached or demand is met.*
$gain = Inf$

Heat only boilers

Increase ***q*** *until boundary is reached or demand is met.*
$gain = Inf$

The contribution of each individual unit type in the EDP method has now been described. When a unit is selected based on its *gain*-value, the code for the corresponding unit is executed. The pseudo code for the whole EDP method is given here below. When a certain unit type is mentioned in the code it refers to the unit pseudocode described above. The EDP method converges when all demand constraints have been fulfilled.

The EDP method:

```
1   Check demand feasibility
2   Calculate best gain for all active units
3   While( not stop )
4       select unit with best gain
5       if( unit == Condensing )
6           do Condensing()
7       if( unit == Heat only boiler )
8           do HeatOnly()
9       if( unit == Back pressure )
10          do BackPressure()
11      if(unit == Extraction )
12          do Extraction()
13      if( all demands are met)
14          stop
15  end while
16  if(production exceeds demand)
17      do repair()
18  end of algorithm
```

Since the EDP method does not allow a decrease in production, except for the lower iso-fuel line in extraction units, it can occur that production exceeds demand for either heat or electricity. This only occurs when production takes place in back pressure units and along the c_m-line in extraction units. If for example heat is no longer needed and the selected *move* is along a c_m-line, excess heat production will take place because this type of move gives both products. After the method converges it checks if excess production can be decreased. This is what the repair function in line 17 of the pseudocode does. In this case the function would make a list of available heat only boilers or extraction units. Heat production is only decreased in extraction units if electricity production can remain unchanged. The function selects the unit with the highest production cost and decreases heat until total production equals demand or no unit is longer available for selection. In the case of excess electricity production, electricity is decreased in condensing units and extraction units (leaving heat production unchanged). A pseudocode for the repair function is given by:

1. ***U*** *:= list of units where production can be decreased*
2. *while(production > demand AND* ***U*** *not empty)*
3.   *find unit with highest production cost*
4.   *take unit out of* ***U***
5.   *decrease production until boundary is reached or demand is met.*
6. *end while*

The construction and implementation of the EDP method has now been discussed in some detail. The method has two main weak points which are difficult to avoid:

- Excess production can occur when moving along the c_m-line in back pressure and extraction units
- The heat production has too much weight in extraction units and the method has a tendency to get stuck in the point (e, q_{max})

The first point is a result of the greedy approach used. It can happen that the back pressure unit with the lowest production cost gives too much of product that is not needed, while a more expensive back pressure unit could give better ratio between products and less overall cost.

The second point is also a result of the greedy approach. Since heat is less expensive than electricity too much weight is put on heat production initially in extraction units. When more electricity production is needed later in the iteration, maximum heat production might have been reached earlier and increase in electricity is no longer available. The search strategy used for the extraction unit is rather crude but in order to make the iteration process not too complicated, it was necessary to make some simplifications. In the next section the performance of individual unit types are tested and the consequences of these approximations will be demonstrated.

3.4 Testing the EDP heuristic

The testing of the EDP method was divided into two steps. In the first step, the performance of the method was investigated for each unit type. The objective with this test is to see how the EDP method performs for each type of unit. These tests should give an idea where the weak and strong points of the method lie. In the second step the EDP method is tested on the whole production system (25 units). The objective here is to measure the performance of the method on the power production system. This test should give an estimate on how the EDP method can be expected to work as a solver when the unit commitment problem is solved. All test results are compared to optimal solutions calculated with GAMS and the error is reported.

3.4.1 Testing individual units

In this section, individual unit types are tested separately. For extraction- and back pressure units both electricity and heat demand is used. For unit types with only one product, the relevant demand is used. Each test is constructed in the following way:

- The number of time periods is 24.

- The number of units in each test is 4 and all units are active in all time periods.
- For each product, the demand is scaled so that the maximum demand is 70-90% of the total production capacity.

The results from these tests will now be presented.

Extraction units

The result for the 4 extraction units are shown in figure 3.4. The upper most figure shows the total electricity production and the figure in the middle the total heat production of the 4 units. The optimal GAMS production is marked with an asterisk ($*$) and the production obtained by the EDP method is marked with a circle (o). The red lines on the plot show the demand. The figure in the bottom gives the error with respect to the production cost for the EDP method. The error is plotted as a function of electricity load demand $l^e(t)$:

$$l^e(t) = \frac{D^e(t)}{\sum_{i=1}^{N} e(i,t)_{max}} \tag{3.2}$$

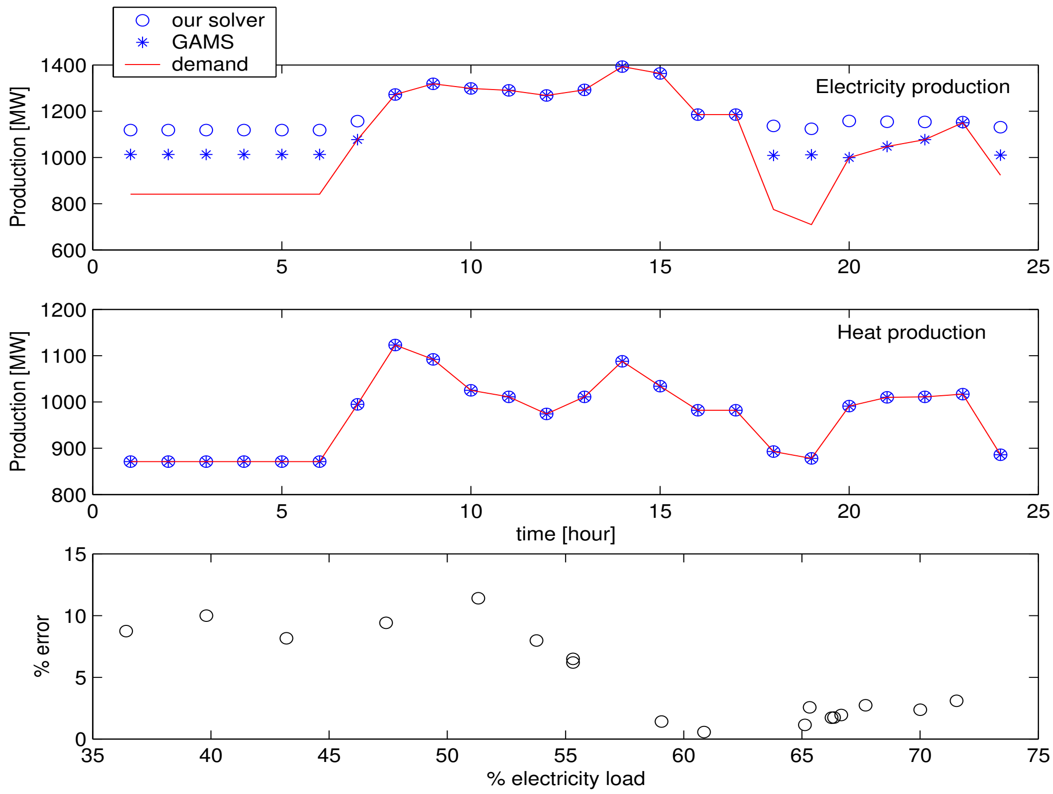

Figure 3.4: Comparison between the EDP method and the optimal solution obtained by GAMS for extraction units. The top most figure shows the electricity production and the figure in the middle shows the heat production. The bottom figure shows the error with respect to the production cost as a function of load demand.

Figure 3.4 clearly shows that best solutions are obtained when the load demand is high. When the electricity load demand is less than 55% the error with respect to production cost is around 10%. For higher load demand the error is 0-3%. The excess production in both solvers when the demand is low, is because the minimum allowed capacity exceeds the demand for those time periods. What is also interesting is that the difference between the optimal heat production and the production obtained by the EDP method, is almost zero for all time periods. It is in the electricity production where the largest differences occur. The reason for this difference is that the cost coefficient for heat $b(i)$ is always lower than the one for electricity $a(i)$ in extraction units. As discussed in last section, the EDP method puts more emphasis on the heat production in extraction units and has a tendency to get stuck in areas where electricity can not be lowered (on the c_m-line, see figure 2.11). To decrease this difference, more emphasis has to be put on electricity production when the initial search direction is chosen. That might however turn the picture around and produce similar effect for heat production.

Back pressure units

Back pressure units have the characteristics that a fixed amount of heat is produced for every unit produced of electricity and vice versa. Since there are two demands that have to be fulfilled and the ratio of electricity and heat can not be controlled within each unit, it was expected that excess of either product would be observed in this test. Figure 3.5 shows the results from this test. Same type of plot is used as in the test for extraction units.

The figure shows that both solvers give excess production of either heat or electricity for almost all time periods. In contrast to the test for extraction units, the largest errors ($\approx 5\%$) are now obtained when the electricity load demand is high. A possible explanation for this is that the EDP heuristic selects units in a greedy way with respect to production cost only. The ratio between heat and electricity is not taken into account. When the demand is low, both solvers choose the same units for production. Excess electricity production can in this case not be avoided. When the electricity demand is high, more units are involved in the production and an optimal solver such as GAMS selects the production pattern based on production cost and the ratio between products. The production is in this case distributed between the 4 units in such a way that excess production is largely avoided. The EDP heuristic considers the production cost only and excess production takes place.

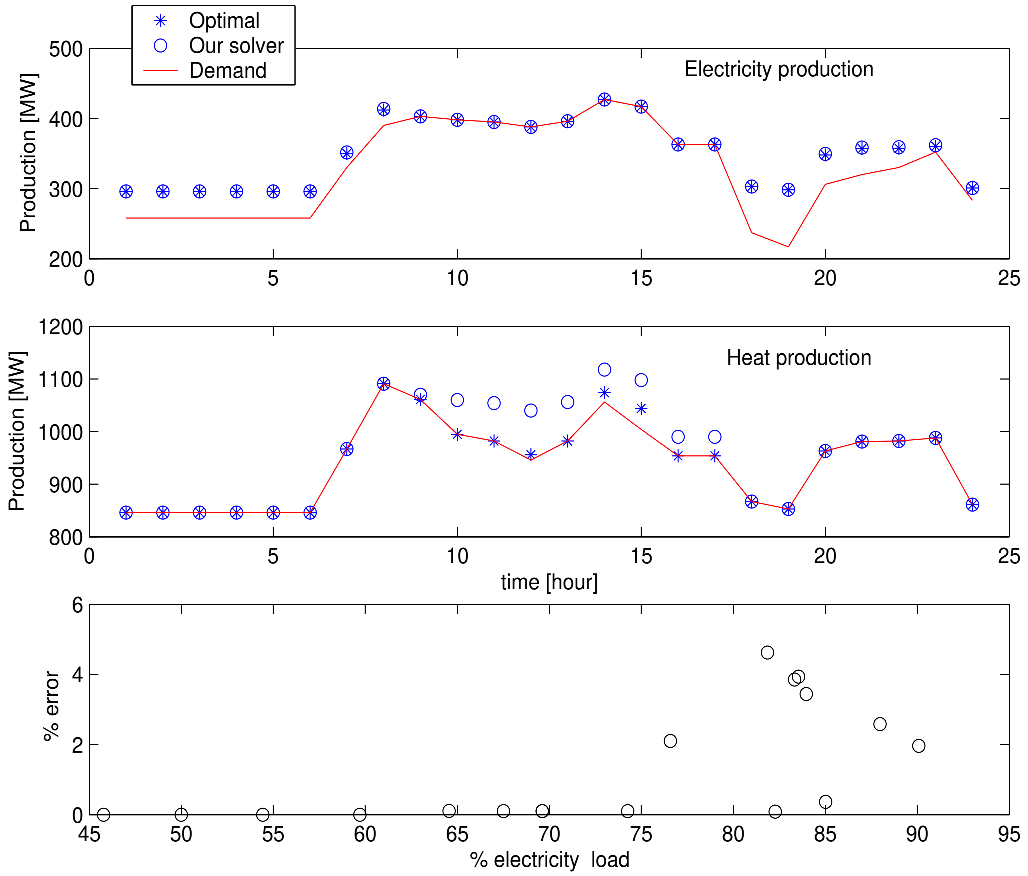

Figure 3.5: Comparison between the EDP solver and optimal solution obtained by GAMS for back pressure units. The top most figure shows the electricity production and the figure in the middle shows the heat production. The bottom figure shows the error with respect to the production cost as a function of load demand.

Condensing units and heat only boilers

Both these units were tested and an optimal solution was obtained in both cases. This is perhaps not so surprising because there is not another product that complicates the overall production. The best possible unit is chosen in each iteration and produces until maximum capacity is reached or demand is fulfilled. The only possible error that could occur is if the EDP heuristic would not choose the best unit during the iteration. Such an error was never observed.

3.4.2 Testing the whole system

The EDP heuristic was finally tested on the whole system which consists of 4 extraction units, 10 back pressure units, 4 condensing units and 7 heat only boilers. The test is made by using a greedy method to iterate on the unit commitment problem repeatedly (without start-up cost). Random unit combinations are generated and the objective function calculated by both solvers. If GAMS estimates a unit combination to be improving, it is accepted. When improvements can no longer be found, a poor unit combination is generated and the greedy iteration is repeated. The results are shown in figure 3.6 where the upper figure shows the results for both the EDP heuristic and GAMS. The lower figure shows the percentage difference in the objective values found by the two solvers.

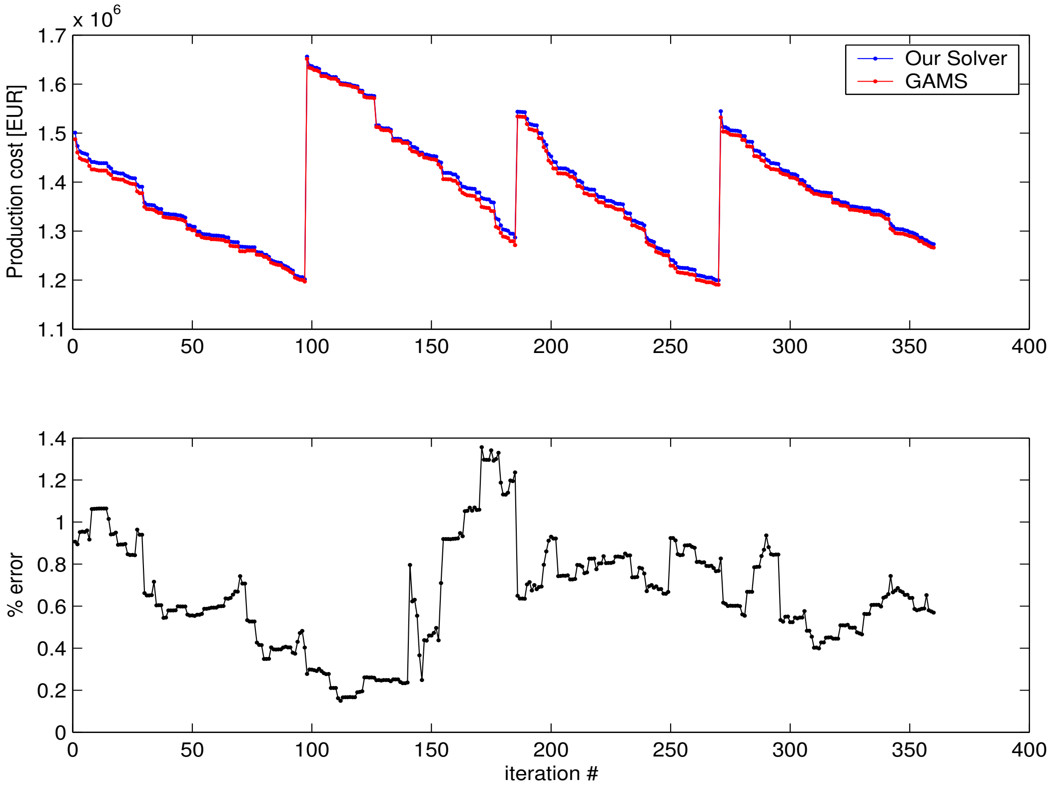

Figure 3.6: The upper figure shows the iteration trajectory from a greedy method used repeatedly. Each point corresponds to one unit combination. The objective function value for both the EDP heuristic and GAMS is plotted. The lower figure shows the % error of the EDP heuristic, compared to the optimal GAMS solution.

The maximum error from these 350 measurements is 1.5% which is a surprisingly good result. The errors in the plot are not completely random but there is still no obvious pattern in the plot. High errors occur both for high and low objective values.

The main result from this test is that the solutions obtained by the EDP heuristic estimate different unit combinations in the same manner as the optimal solver. This experiment indicates that if in two successive iterations k and $k+1$, the optimal solution from GAMS has the property that $f^*(x_k) > f^*(x_{k+1})$, the EDP heuristic estimates the change in the same way $f(x_k) > f(x_{k+1})$. This result is very important and indicates that the EDP heuristic can be used to estimate the quality of different unit combinations in a similar manner as an optimal solver. It is however difficult to prove if this holds for all possible cases.

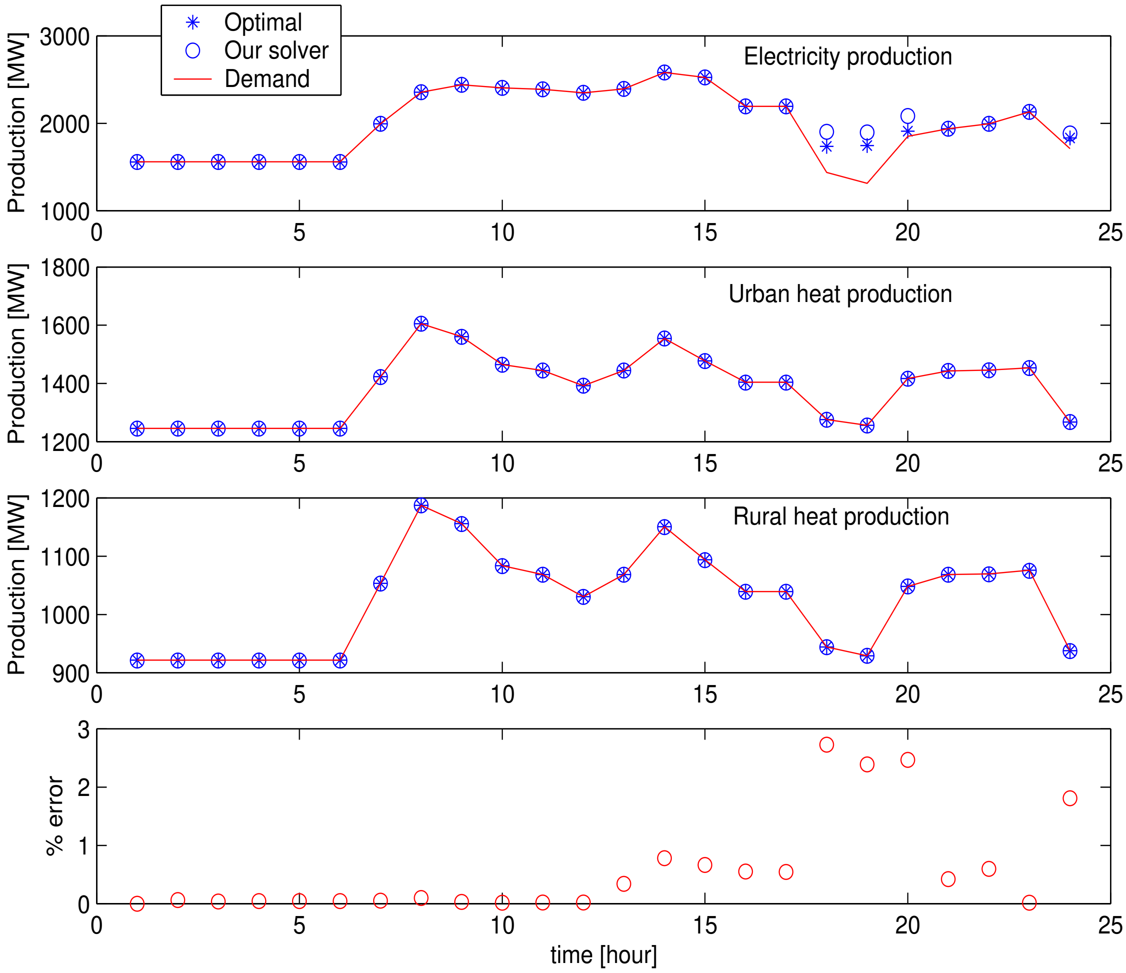

Figure 3.7: The figure shows an example of solution from the EDP heuristic for the whole system. Electricity, urban heat and rural heat production is plotted. The error with respect to the optimal production cost is given in the bottom figure.

Figure 3.7 shows the solution obtained by the EDP heuristic for the final unit combination in figure 3.6. This figure shows the same kind of plot as shown for back pressure and extraction units before. The only difference is that the model formulated in chapter 2 is now solved, which includes urban and rural heat production. The bottom figure shows the error compared to the optimal solution (with respect to the objective function). The figure shows that the production from both solvers produces very near to demand in almost all time periods. The one exception is in time periods 18-20, when the demand decreases suddenly. In that case too many units are open and electricity production exceeds demand. The maximum error that occurs in one time period is 3% which is quite promising.

The test shown in figure 3.7 was done repeatedly when the EDP heuristic was tested after implementation, using thousands of different unit combinations. The maximum error observed was always less than 4%. Even though a system of individual unit types can produce results with up to 10% error, the errors from different units seem to compensate each other when the system is composed of several units. This is exactly what is observed in figure 3.6 where each error in the plot is an average over 24 time periods and is always lower than 1.5%.

Time measurements

Finally the EDP heuristic was tested with respect to computation time. The following results were obtained:

- Solving the whole power production system (25 units) over 24 time periods takes on average **1.2msec**. For one time period the average time is **0.048msec**
- The average number of iterations is 11 per time period and the average increase in production per iteration is around $280MW$

These time measurements are extremely important since a computation time of 1.2msec is quite satisfactory for such a large system. This should give flexibility with respect to computation time for the method that solves the combinatorial part of the UCP. The number of iterations needed is quite small as could be expected since the EDP is linear and advantages of that is taken in the calculation.

3.5 Conclusion

The subject of this chapter has been the construction and implementation of a heuristic method that solves the economic dispatch problem. Initially the use of a linear solver was tested but it was concluded that such an approach would be too slow. The heuristic method implemented, was then described on a general basis in section 3.2 to give an idea about the principles behind the method. The detailed discussion in section 3.3 was made to give more specific implementation details for the interested reader. The heuristic was then tested both for individual unit types and on the power production system presented in chapter 2

Based on the test results, it is concluded that the EDP heuristic generally performed quite well for the power production system analyzed. The maximum deviation from an optimal solution that was observed was less than 4% for one time period. The average over 24 time periods was always less than 2%. Error for individual unit types can however be as large as 10% but such high errors seem to balance out when more unit types are used.

An important result from these tests is that the property seems to hold that if an optimal solver estimates $f^*(x_k) > f^*(x_{k+1})$ then the EDP heuristic also estimates

$f(x_k) > f(x_{k+1})$ where x_k is some unit combination for the whole system. This property is very important and indicates that this heuristic is suitable to estimate the quality of different unit combinations. Another important result is that the computation time is only *0.048msec* per time period. It was expected that the method that solves the combinatorial[8] part, would be more limited by the computation time of the EDP heuristic.

In sections 3.2 and 3.3 it was mentioned that this heuristic was constructed in such a way that it could be extended to convex non-linear objective functions. This should be possible by decreasing the step size in each iteration. Instead of increasing production until demand is met or boundary is reached the step size Δe_i and Δq_i would perhaps be $5 - 10MW$. The main modifications to the heuristic that would have to be made, are most likely in the extraction units were e and q can be changed independently. The choice of a search direction would have to be looked at more carefully. This would not be a problem for the other unit types, since there is only one search direction possible.

It would be desirable to be able to work with non-linear cost functions since the production cost would then be dependent on the production status. When the production cost is assumed linear, production cost at full capacity and minimum capacity is the same. In reality, this is highly unlikely and one would expect that fuel consumption increases as the production gets close to the full capacity. Non-linear production cost should therefore give a more accurate picture of the production but at the expense of increased computation time. A lack of time prevented this modification to the EDP heuristic. Given enough time, it would have been the next objective.

The results of using the heuristic presented in this chapter to solve the EDP and Simulated Annealing to solve the combinatorial part of the UCP are given in chapter 6. The subjects of next chapter are the possible solution methods for the combinatorial part and why Simulated Annealing was chosen.

[8] which is Simulated Annealing in this case

Chapter 4

Overview of Solution Methods for the UCP

Last chapter presented a heuristic approach to solve the economic dispatch problem which is the linear part of the UCP. The solutions obtained by that heuristic approach will be used to distinguish between the qualities of different unit combinations when the UCP is solved in chapter 6. After completing the implementation of the EDP heuristic, a decision had to be made on which solution method to choose to solve the combinatorial part of the UCP. There is a wide variety of methods available for selection but they differ in the solution quality they obtain, computational efficiency and size of the problem they can solve. This chapter presents an overview of some of the methods that have been used to solve the UCP and are reported in the literature [19], [20]. The solution methods discussed are divided into three groups:

- Exact methods
- Heuristic methods
- Relaxation methods

Exact methods as they are defined here are methods which can be guaranteed to converge to the optimal solution. Solution methods that are not guaranteed to converge to the optimal solution but do provide good solutions in a reasonable computation time are classified as heuristic methods. Relaxation methods are solution methods that relax certain problem constraints in order to make the problem more readily solvable. The final solution obtained by relaxation methods is not guaranteed to be feasible with respect to the original problem.

This chapter is divided into 3 sections where the methods belonging to each group are discussed. A description of each method is given along with its advantages and disadvantages in connection with the UCP. At the end of this chapter the argument for the selection of Simulated Annealing as a solution method is given

4.1 Exact methods

The obvious advantage of using exact methods to solve the unit commitment problem is that they converge to the optimal solution. Because the decision variables of the unit commitment problem are binary variables, finding the optimal solution is in practice extremely difficult. The reason for this is that the solution space, which consists of all feasible unit combinations, increases exponentially with the number of units and time periods in the model. To find the optimal solution to the problem, the solution space must be searched systematically. In many cases the number of feasible solutions that the methods must go through to converge, is so large that the computation time becomes impractical. The threshold where these methods fail is dependent on the solution method itself and to some extent on the computer system used. Because of the exponential size of the solution space, the unit commitment problem can not be solved by these methods for many practical power production systems. There are 3 different solution methods discussed here that have been used to solve the unit commitment problem to optimality. They are: *Extensive numeration*, *Dynamic Programming* (DP) and *Branch and Bound* (B&B). A short description of each method will now be given.

4.1.1 Extensive numeration

This method is by far the most simple of the three methods mentioned above with respect to implementation. The solution approach used is to generate all possible unit combinations in a systematic way and set aside all feasible combinations with respect to demand and spinning reserve. For each feasible unit combination, the economic dispatch problem is solved and start-up cost is added. The combination that gives the lowest production cost is then selected.

Since all possible combinations of units are tested, this method is guaranteed to find the optimal solution. The obvious drawback of this method is that it only works for small problem instances. The number of unit combinations that needs to be generated increases exponentially with the number of units and time periods. For that reason the computation time quickly becomes impractical. This method is only practical for problems involving small number of units and time periods [19]. Such small problems are however usually of limited interest.

4.1.2 Dynamic Programming

In Dynamic programming (DP) the optimal solution is found recursively by subdividing the problem into T different *stages*. Each time period is regarded as a *stage* and for each *stage* there are n different *states*. The number of *states* in each *stage* is equal to the number of feasible unit combinations [19].

The iteration process can either be made in forward direction starting at *stage* 1 or backward direction starting at *stage* T. In forward iteration the optimal solution in iteration 1 is found by determining the best unit combination for time period 1. The optimal solution in *stage* 2 is then found based on the optimal solution found in *stage* 1. The iteration continues in this fashion until *stage* T is reached.

The principle that dynamic programming is based on is that given a current *stage* i, the optimal decision made at *stage* $i+1$ depends only on the decision made at *stage* i and is independent on the decisions made in the previous stages, $1 \dots i-1$. The formulation of the unit commitment problem must have this property (*Markovian property*) if the method is to converge to optimality. Further reading about DP can be found in [6], [25].

The advantages with dynamic programming compared to extensive numeration is that the number of solutions that must be examined is much smaller and is only dependent on the number of units and not on the number of units and time periods. The number of *states* is however quite large even for small systems since it increases exponentially as the number of units grows. Because of this exponential property, dynamic programming can only be used for small problem instance. Another disadvantages with dynamic programming is that minimum up and down constraints are quite difficult to handle [19].

4.1.3 Branch and Bound

The principle behind the branch and bound (B&B) method is to subdivide the original problem into smaller sub-problems which are easier to solve. The solution found for each sub-problem is then used to limit the search space. For integer problems the division is made in such a way that each sub-problem is the same as the original problem except that certain number of integer variables have been fixed. Figure 4.1 shows how a binary problem is divided into sub-problems. The root of the binary tree is the original problem. The root is then divided into two smaller sub-problems by fixing one binary variable $u(i,t)$, either as 1 or 0. Each of the sub-problems can then be divided into two more sub-problems by fixing another binary variable. For each subdivision the decision tree grows in size until the total enumeration tree has been built.

Instead of building the whole decision tree, certain parts of the solution can be systematically discarded from the search. This is done by relaxing the binary constraint on all variables that have not been fixed and solve the resulting linear problem [1]. The solution obtained by this relaxation has the property that it is always lower than or equal to the solution of the original binary problem (for minimization problems). This is because the solution space for the linear problem includes all feasible solutions for the binary

[1] this means that instead of restricting $u(i,t) \in \{0,1\}$, it can now take values in the range: $0 \leq u(i,t) \leq 1$

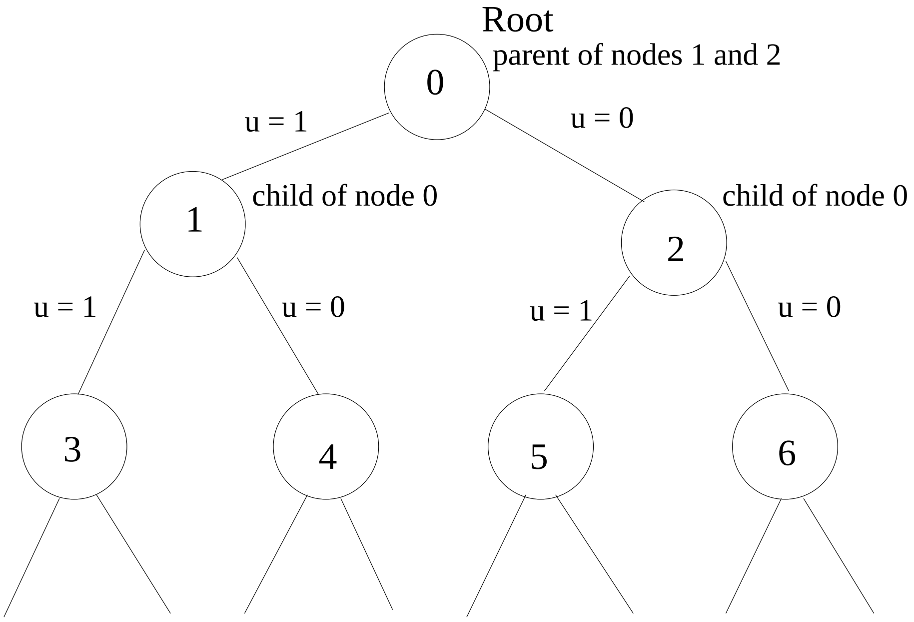

Figure 4.1: The figure shows the construction of the branch and bound search tree. In each node a constraint is imposed on a given binary variable ($u(i,t)$) that its value is either 0 or and 1. This results in two child nodes from the given parent node.

problem and all fractional solutions which can give better objective value than a pure binary solutions. This bound is then used to discard certain parts of the solution space in the following way:

In the first iteration a feasible solution for the root (node 0) is found for example by using heuristic. This solution is called the incumbent solution. If the lower bound estimated for the root node is a feasible binary solution the optimal value is found. Else the root node is divided into two sub-problems, lower bounds estimated and the node with the best lower bound is usually selected. If the lower bound for that node gives a higher solution than the incumbent solution, that node and all its decedents can be discarded from future search[2]. If not, the method tries to find a feasible binary solution for this node. This is for example done by using cutting planes algorithms [25] . If a binary solution is found, it is compared to the incumbent solution and replaces that solution if its value is lower. If a binary solution can not be found or the node can not be discarded based on its bound, the node is divided into two more nodes. The best node from the set of active nodes is then chosen and the iteration continues. It is usually easier to find feasible binary solution to a given node as the number of fixed variables increases. The algorithm converges when the best incumbent solution found can be used to exclude all other nodes in the tree.

[2]if a lower bound for node i gives worse value then the incumbent binary solution, the best binary solution for node i can obviously not give a better solution than its bound. Hence the node and all its descendents can be discarded from the search

This is the principle that the B&B method is based on. Different B&B algorithms differ in what order the nodes in the tree are divided and what variables are chosen to be restricted to binary values. Further discussion on the B&B method can for example be found in [25].

Because of the ability of B&B to exclude parts of the solution space in a systematic way, the method has been quite successfully applied to many medium sized UCP problems [2], [5] . The method is quite extensively used in many commercially available solvers for mixed integer programming (MIP) problems and these solvers are becoming more and more sophisticated. Branch and Bound is for example used when MIP problems are solved in GAMS[3]. For large problem instances the size of the binary tree becomes too large and the tree can not be searched through in a reasonable computation time.

4.2 Relaxation methods

For many problems there are one or more constraints which make the given problem "difficult" to solve. The word "difficult" here means that if these constraints are dropped from the original formulation the resulting problem becomes easier to solve. Instead of dropping these constraints completely from the formulation it is also possible to incorporate them into the objective function such that penalty is paid when they are not satisfied. Methods that use this strategy for solving problems are classified here as relaxation methods.

The solution methods discussed here are Linear Programming (LP) and Lagrangian relaxation (LR). Both methods solve a version of the UCP where constraints that are "difficult" for each method have been relaxed. This is practically the only thing that these two methods have in common. Linear programming should actually be classified as an exact method since it produces optimal solution for the problem it solves. Since it solves the relaxed version of the UCP it is placed in this section. The solutions that these two methods produce with respect to the original problem are at best optimal but in some cases the they produce sub-optimal or even infeasible solutions.

4.2.1 Linear Programming

Since the UCP has binary decision variables, it can not be solved directly by linear programming (LP). However by relaxing the binary constraint this method can be used. This effectively means that the minimum capacity in each unit is set equal to 0 and the resulting problem is simply the economic dispatch problem. Because of the continuous capacity from 0, start-up costs can not be used and unit selection is purely based on

[3]for linear MIP problems the solver that GAMS uses is CPLEX which has a B&B routine.

the production cost. Figure 4.2 shows the resulting working area for back pressure and extraction units after the LP relaxation.

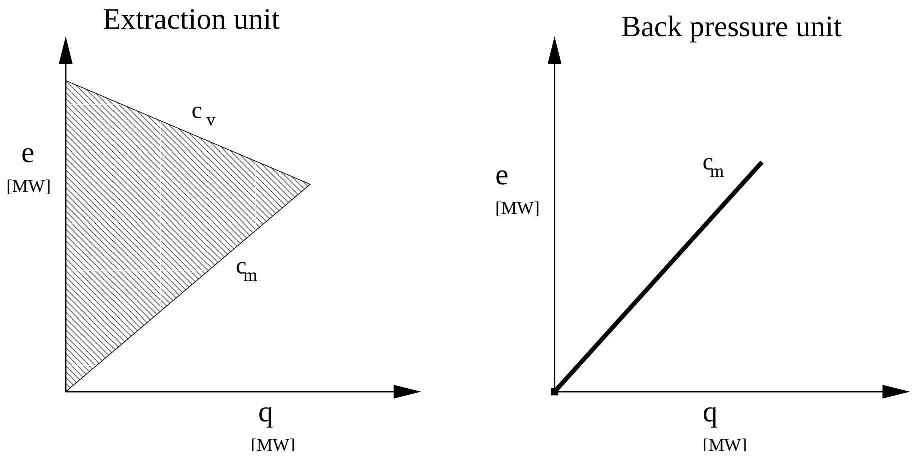

Figure 4.2: The figure shows the resulting working area for back pressure and extraction units after the LP relaxation.

For the UCP, the LP relaxation is in most cases a very crude approximation since it is highly likely that the solution to the relaxed problem is infeasible with respect to the original problem. This method is however feasible to use when the number of units is large or the time horizon under study is long. This is for example the case in the *Balmorel* model. The number of units used there can be up to a few hundred and the time horizon can range from days and up to several years. In such cases linear programming is extremely stable compared to other methods since large linear problems can be solved quite easily. When the linear problem becomes too large, decomposition methods have been used[4] [19].

Linear relaxation has been used for simulating the CHP system in the Copenhagen area and the results have been compared to optimal solution without relaxation [5]. The difference in the objective function value was reported to be less than 1% but final solution had to be changed manually to reach a feasible solution.

4.2.2 Lagrangian Relaxation

In the UCP, time periods are bound together by the start-up cost and for a given time period all units are bound together by the demand and spinning reserve constraints. The idea behind using Lagrangian relaxation (LR) for solving the unit commitment problem is to subdivide the solution space into smaller areas by relaxing the demand and spinning reserve constraints. Instead of dropping the demand and spinning reserve completely (which would make the problem meaningless) the constraints are embedded in the objective function and penalty multipliers are assigned to each constraint. The resulting objective function can be rearranged to create N sub-problems, one for each

[4] such as Dantzig-Wolfe decomposition

unit, where each sub-problem is only dependent on the number of time periods and independent of other units. To reach the constraint optimum, the multipliers are updated by solving the dual problem of the original master problem. The dual problem has lower dimension than the original problem and is easier to solve.

In order to demonstrate how this method works mathematically, a UCP, which consists pure electricity production without spinning reserve constraint, is chosen to save space[5]. All other constraints are assumed to be included as discussed in chapter 2 and the objective function (Z) is the same as shown in equation 2.23 with the exception that the heat variable is omitted.

The Lagrangian function is then given by:

$$L(e,u,\lambda) = Z(e,u) + \sum_{t=1}^{T} \lambda(t) \Big(D^e(t) - \sum_{i=1}^{N} e(i,t) \Big) \tag{4.1}$$

$$\lambda(t) \geq 0 \quad \forall t \tag{4.2}$$

Optimizing with respect to this function, subject to the relevant constraints, is done in two steps. In the first iteration initial values are assigned to the Lagrangian multipliers and the master problem is solved. The master problem consists of minimizing $L(e,u,\lambda)$ with respect to e and u for a fixed value of λ.

The Master problem

$$q(\lambda) = \min_{e,u} L(e,u,\lambda) \tag{4.3}$$

The λ multipliers are then updated by solving the dual of the master problem where e and u have the values previously found by the master problem and λ is a variable:

The dual problem

$$q^* = \max_{\lambda} q(\lambda) \tag{4.4}$$

The Lagrangian multipliers are then sent back to the master problem and the iteration continues. The difference of the objective value between the master and the dual problem gives the duality gap. The duality gap provides a measure of the near optimality

[5] otherwise 4 demand and spinning reserve constraints would be embedded in the objective functions

of the given solution. The iteration process is usually continued until the duality cap is less than some tolerance limit. More general discussion about Lagrangian relaxation can be found in [25].

As mentioned before the master problem can be subdivided into N sub-problems (one for each unit). This is done by rearranging the terms in equation 4.1:

$$\min_{e,u} L(e,u,\lambda) = \sum_{i=1}^{N} \min_{e,u} \sum_{t=1}^{T} [z_i(e(i,t),u(i,t)) - \lambda(t)e(i,t) + \lambda(t)D^e(t)] \tag{4.5}$$

where $z_i(e(i,t),u(i,t))$ is the production cost in unit i.

The master problem is usually solved using dynamic programming or branch and bound and the dual problem can be solved with sub-gradient methods [19]. When the problem size is large heuristic methods have to be used to update the Lagrangian multiplier. Genetic algorithms have for example been used quite successfully in this connection [3, 4, 14, 24].

Compared to other methods, the advantages of using LR for solving the UCP is that the method is quite flexible in handling different unit constraints and is quite efficient with respect to computation time [19]. The number of units this method can handle is relatively large but the length of the time horizon is more sensitive because of the way the sub-problems are solved (DP or B&B). This method has been applied successfully on large practical power systems using thermal, hydro and fuel constrained units [22]. The main disadvantage with the LR method is that the final convergence of the Lagrangian multipliers can be problematic. This can lead to infeasible final solutions with respect to demand and spinning reserve constraints. Using quadratic penalty multipliers has reduced this problem somewhat but the final convergence of the λ's is still problematic.

4.3 Heuristic methods

The word "heuristic" derives from the Greek word "heuriskein" and means "to discover" [27]. A heuristic method is a well defined sequence of steps that identifies a high quality solution in a reasonable computation time for a given problem. Optimality of the final solution can not be guaranteed and in many cases it is not possible to estimate how close to the optimum the final solution is. The fact that optimality can often not be verified or guaranteed has its pros and cons. The exact methods discussed in section 4.1 guarantee optimal solutions but the price is often a large amount of computation time used to improve solutions that beforehand are quite close to the optimal solution. This is for example the case with B&B which often finds a near optimal solutions in a short computation time. During iteration it can happen that the computer memory can not handle the size of the B&B tree and the algorithm breaks down before converging.

As a result the "good" solution already found is lost. For heuristic methods optimality is not given such a high priority but the emphasis is on finding good solutions in a short time. This often results in the solution method being more simple and transparent than exact solution methods with respect to implementation.

Heuristic methods are used extensively in the field of operation research, specially in connection with solving combinatorial optimization problems. The popularity of heuristic methods is among other things related to the following properties:

- They often give good solutions in short computation time where exact methods either fail to produce a solution or are too slow with respect to time.
- They are often more simple to implement than exact methods.
- They are often less sensitive to the model formulation than exact methods and can handle more difficult objective functions and constraints.

The solution strategy that different heuristic methods use can differ significantly. Some methods are curtailed specifically around a certain type of problem, for example like the EDP heuristic in chapter 3, while some methods are more general such as metaheuristics.

The word metaheuristics is used for a set of heuristic methods that can be applied to a wide set of different problems. These methods can be described as a general algorithmic framework which can be adapted to a specific optimization problem with relatively few modifications. Examples of metaheuristic methods are *Simulated Annealing* (SA), *Tabu search* (TS) and *Genetic Algorithms* (GAs). All these methods have been used to solve the unit commitment problem and have worked quite well in practice. These methods will now be discussed in some detail. In addition, two other heuristic methods, *Priority List* and *Steepest Descent* will also be discussed.

4.3.1 Priority list

The priority list is one of the most simple methods used to solve the unit commitment problem. In this method a priority list of units is created based on their estimated production cost. The function that estimates the production cost is called the priority function. Different cost factors such as fuel and start-up costs are given different weights so the priority function can give a realistic estimate of the production cost in each unit. When the production cost in each unit has been estimated the units are listed in increasing (or decreasing) order based on the estimated production cost. Units are then selected from the list one by one until enough capacity is reached to fulfill demand and spinning reserve constraints.

There is obviously a fundamental difference between this method and the other methods discussed here. All the other methods use the solution from the economic dispatch problem to determine the feasibility of each unit combination. Here however, the production cost in each unit is determined independent of the other units. The advantage of this method is that it is extremely simple and easy to implement. It does however produce sub-optimal solutions which can sometimes be of poor quality. This method is often used as a first estimate in unit scheduling or to find a good starting point for other heuristic methods [19].

4.3.2 Steepest descent

Steepest descent is like priority list a rather simple heuristic. The reason for its discussion here is to prepare for the discussion of Simulated Annealing and Tabu Search in the next two sections. The solution strategy used by both methods is strongly related to this heuristic.

The steepest descent method belongs to a class of so called local search heuristics. The use of this class of methods to solve combinatorial problems is quite popular. For local search methods, each newly selected solution x_{k+1} in the solution space S is generated in the neighborhood $N(x_k)$ of the current solution x_k and selected by some acceptance criteria. The definition of a neighborhood solution usually refers to a solution that is generated by modifying the current solution slightly. For the unit commitment problem the neighborhood to a current solution $N(x_k)$ could for example be all solutions generated by changing the on/off status of one unit over some limited time period.

The acceptance criteria that the steepest decent method uses, is to accept only solutions that give lower function value than the current value (for minimization problems). The method starts with an initial feasible solution and in each iteration the best solution in the neighborhood of the current solution is chosen. This is repeated until no more improvements can be made. A pseudocode for the steepest descent method is as follows:

Initialize $k = 0$
Select an initial feasible solution $x_0 \in S$
    while(improvements can be made)
        select best $x_{k+1} \in N(x_k)$
        if($f(x_{k+1}) < f(x_k)$ *)*
            replace x_k *by* x_{k+1}
        $k = k + 1$
    end while
return x_k

When this method is implemented it is in most cases not practical to explore the whole neighborhood $N(x_k)$ and select the best solution. Instead certain number of solutions n are generated and the best one is selected. In that case the word "steepest" is usually omitted when referring to this method.

The number of solutions n generated in $N(x_k)$ has a significant effect on the performance of the method. Large value of n usually speeds up the convergence and reduces the variability between different solutions. The reason for this is that larger neighborhood increases the probability that the same "good" solutions are selected each time the method is used. Larger neighborhood search does however not guarantee better final solution. Even though the selection of the neighborhood size is important for the final solution, the most important factor for the quality of the final solution is the definition of the neighborhood structure itself. This factor is discussed in more detail in chapter 5. Further discussion of steepest descent can be found in [17].

The main advantages of using steepest decent is that it is easy to implement. Its main disadvantage is that it is easily trapped in local minima and the quality of the final solution is highly dependent of the initial starting solution. Both these properties are due to the fact that the method never accepts a "bad" solution and can therefore not escape local minima. To overcome this problem more sophisticated neighborhood search methods have to be used. Two of these methods are discussed in the next two sections.

4.3.3 Tabu Search

Tabu search (TS), like steepest descent, is based on neighborhood search strategy. A crucial difference between the two methods is that Tabu search has a mechanism to climb out of local minima. This is done by keeping a dynamic list of recently visited solutions and forbidding all movement back to these solutions for a certain number of iterations. The length of the tabu list l is implementation dependent and depends on the problem being solved. In the first l iterations all solutions are added to the tabu list. After that each new solution is inserted into the list and the oldest value in the list is discarded. When a local minima is reached during iteration the current solution is put into the list and the same solution can not be used in the next iteration. A worse solution must therefore be selected in the next iteration. If the tabu list is too short, the influence of the latest visited local minima might not have disappeared which can result in the solution circling back and forth in that local minima. If properly implemented the method "walks" back and forth around the solution space visiting several minima. The best solution seen so far is kept. The iteration process usually stops when some maximum number of iterations is reached or the number of iterations since the best solution was improved has reached some specified limit.

The factors that are most important for the performance of Tabu search are the size of the neighborhood search, the neighborhood structure and the implementation of the tabu list. The first two factors have already be mentioned or will be discussed in chapter 5. In most cases the tabu list does not use the exact solutions to determine which solutions are allowed. Usually the list holds information about some specific attributes of the recently visited solutions. The reason for this is that it can take too much memory and time to store a number of complete solutions. In the case of the UCP, one solution consist of a $N \times T$ matrix and storing many such solutions uses a lot of computer memory if N and T are large. It is also time consuming to verify if two solutions are identical. In the attribute approach it could for example be enough to store information about which units have been recently modified. The attribute approach can have the result that solutions not in the tabu list become forbidden for some number of iterations. This can help the algorithm to climb out of local minima but it can also exclude some "good" solutions from being explored. More detailed description on Tabu search can be found in [7], [17].

Tabu search has been used to solve the UCP and the reported results have been promising [12]. The main advantages with this method is that it is relatively easy to implement. Compared to methods such as LR, DP or B&B, the iteration process in tabu search is also much more transparent. The method can handle large problem instances and does not break down because of the size of the solution space as exact methods do. As with most heuristics, larger problem instances do however affect the quality of the final solution and the computation time. The main disadvantage with this method is the computation overhead around keeping the tabu list.

4.3.4 Simulated Annealing

Like Tabu search, Simulated Annealing (SA) is also based on local search strategy and has a mechanism to climb out of local minima. The main difference between the two methods is how this mechanism is implemented. The iteration process for SA is very similar to steepest descent with one major difference. Instead of accepting only solutions that give lower function value than the previous solution, SA accepts non-improving solutions with probability p. The acceptance probability depends on a control parameter T, and the difference in the objective value of the two solutions $\Delta = f(x_{k+1}) - f(x_k)$. This acceptance probability gives the possibility of jumping out of local minima and potentially fall into a more promising downhill trajectory. By modifying the acceptance probability in a controlled manner the idea is that the iteration trajectory reaches equilibrium in some "good" areas in the solution space and converges. Unlike Tabu search, SA draws only one solution at a time from the neighborhood of the current solution $N(x_k)$.

The probability distribution used for accepting worse solutions is usually the Maxwell Bolzman distribution $e^{-\Delta/T}$. The control parameter T is referred to as the temperature. In the first iteration the value of T is set large enough so that the probability of acceptance is close to 1. This allows the solution space to be explored and decreases the effect of the initial starting point. As the iteration progresses T is decreased slowly until the acceptant probability is close to 0 and only downhill steps are chosen. Further information about SA can for example be found in [15], [25].

As for tabu search, simulated annealing has been used to solve the UCP with good results [13, 26, 28]. The advantages of using simulating annealing are almost the same as those mentioned for Tabu search. The method is easy to implement, its solution trajectory is transparent and it can handle large problem instances. Another advantage is that it does not need much computer memory. The solution quality obtained by SA depends on the cooling schedule and the neighborhood structure used. The cooling schedule is in most cases relatively easy to tune. The most difficult part is to implement the neighborhood structure. The main disadvantages with SA is that in many cases the temperature must be reduced slowly which can result in large computation times.

4.3.5 Genetic Algorithms

Genetic algorithms (GAs) are adaptive search techniques which are based on the principles of natural selection. The method operates as an iterative search procedure on a set of candidate solutions of fixed size (called population). Each candidate solution is usually encoded as a binary or integer string and the encoded solution is called a chromosome. The quality of each chromosome is estimated by a fitness function which usually is the original objective function or some function which increases (or decreases) monotonically with the objective function. In the iteration process better and better set of solutions are produced from the original population by using selective breeding and recombination strategies. Each set of population is referred to as a generation. The two main genetic operators used to generate new solutions from the current population are the mutation operator and the crossover operator.

- *Crossover*
  Two chromosomes from the current population, called parents, are combined (exchange bits) to form two new solutions (offsprings). The new chromosome inherits solution characteristics from both parents.

- *Mutation*
  With a small probability, randomly chosen bits of offsprings are changed (from 0 to 1 or vice versa if the solution string is binary). Mutation injects new properties into the current population and prevents the algorithm to get stuck in a local minimum.

An example of both crossover and mutation can be seen in figure 4.3

After generating the initial population (the first generation), the iteration process consists of the following steps:

1. **Evaluation**. The fitness of individual chromosomes in the generation is evaluated.

2. **Parent selection**. Certain pairs of chromosomes (parents) are selected based on their fitness value.

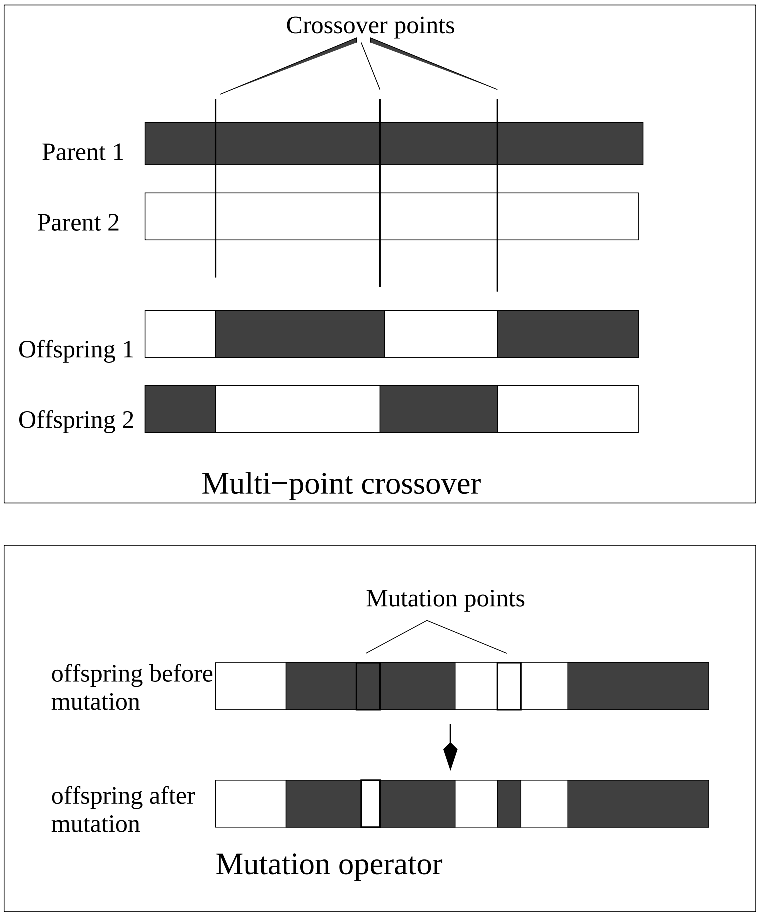

Figure 4.3: A schematic representation of the crossover operator (multi-point)and the mutation operator.

3. **Crossover**. Each pair of parents combines to produce one or two new chromosomes (offsprings).

4. **Mutation**. The mutation operator is applied with certain probability to some of the offsprings.

5. **Population selection**. Based on their fitness value a new population is selected replacing some or all of the original population by an identical number of offsprings. This completes the formation of a new generation of candidate solutions.

This procedure is then repeated until the chromosomes in the population converge. Many generations are needed for convergence to a near optimal solution. Further information about GAs can be found for example in [25]

Genetic algorithms have been used quite extensively in solving the unit commitment problem [8, 19, 21]. The method is quite effective in finding good areas in the solution space but it is however not as effective in finding good solutions locally inside that area [19]. To overcome this problem hybrid methods of GAs combined with other heuristics have been used with some success [1], [11]. Other methods such as Lagrangian relaxation have also been used with GAs and produced good results [3, 4, 14, 24].

One reason for the popularity of GAs for solving the unit commitment problem is that it can be used for quite large power production systems and it is rather simple to represent its solution in a form of a chromosome (binary string). Figure 4.4 shows how a solution for the UCP with N production units and T time periods might be presented. Normally the unit schedule is presented by a matrix where the unit status in each time period is given in the columns of the matrix. When this solution is encoded into a chromosome the unit scheduling is represented in one long vector as shown in the figure.

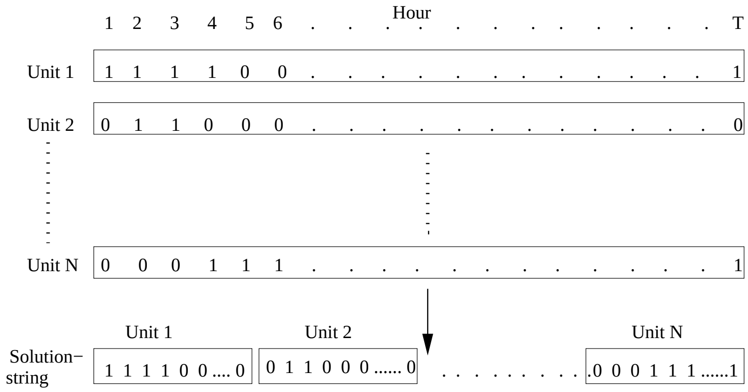

Figure 4.4: Binary representation of a UCP solution. 1 means that unit is active, 0 means that unit is not active.

One disadvantages of using genetic algorithms for solving the UCP is that it is difficult to maintain feasibility for the minimum up and down constraints. The reason for this is that feasible schedules are frequently destroyed when either the mutation or the crossover operators are applied. There are two ways to overcome this problem. One way is to use penalty factors in the fitness function to guide the mutation and crossover to produce feasible solutions. Another way is to implement a repair mechanism to fix infeasible solutions after the operators have been applied. Compared to SA and TS, the implementation of genetic algorithms is more complicated and the method is more difficult to tune. Because many candidate solutions are used simultaneously, the iteration trajectory is also less transparent. It has been shown [19] that the computation time of GAs increases in a quadratic way with number of units in the system. This time can be somewhat reduced by using parallel implementation or concurrent processing. That does however make the implementation of this method much more complicated.

4.4 Conclusion

The purpose of this project is to solve the unit commitment problem using heuristic methods. The reason for using heuristic rather than exact methods was the problem instance used here, was on the verge of being so large that exact methods could not be used. Another reason was that a similar type of problem had been studied before in a another master thesis [5]. The problem under consideration there concerned the CHP units in the Copenhagen area, which consisted of 15 different generation units. Linear relaxation and branch and bound were used to solve the problem and both methods were implemented in GAMS[6]. Instead of repeating that work for a slightly different system it was both more logical and interesting to test the performance of a heuristic method for this system. This has the advantage that the solution method could easily be extended to larger system.

Based on the above assumption, the methods of choice could be reduced to choosing between Lagrangian relaxation, Genetic algorithms, Tabu Search and Simulated Annealing. All these methods have been used to solve the UCP and all have been reported to give near optimal solutions for fairly large systems. Lagrangian relaxation was excluded for the simple reason that it is difficult to implement properly. The convergence of the Lagrangian multiplier is problematic for large systems and finding a suitable method to overcome that problem would be a whole study in itself. It had already been found out that solving the economic dispatch problem with exact methods would be problematic which led to the development and implementation of the EDP heuristic discussed in chapter 3. The three metaheuristic methods GAs, TS and SA are better suited than LR to use with that solver. Of these three methods GAs is the most difficult to implement and the solution process is less transparent compared to the other two methods. The implementation difficulty was not considered to be an important factor but the transparency of the solution process is more important. This is manly because the EDP solver had just been implemented and it is easier to correct, modify and test the solver using a simple and transparent method such as SA or TS. The choice between SA and TS was made arbitrarily and I chose to implement SA, while my partner chose to implement TS. It turned out that the transparency of Simulated Annealing[7] was essential to overcome the final flaws in the implementation of the EDP-solver.

When testing heuristics, it is always desirable to be able to compare the results to an optimal solver if such a solver can be used. If such a solver can not handle the problem then a smaller test problem can be used to check the performance of the heuristic method. The B&B method is available in the GAMS programming environment both for linear and non-linear problems. The GAMS implementation of the UCP as presented in chapter 2 is not difficult if one is familiar with the GAMS syntax. Since this method was available it was decided to implement the problem in GAMS, to give comparison with the SA/EDP heuristic. The GAMS code can be found in appendix C.2.

[6]Linear relaxation was used when the problem size became too large for B&B
[7]same holds for Tabu Search

Chapter 5

Implementation of SA

The subject of this chapter is the implementation of SA. Before going into implementation details, the first part of this chapter is devoted to the background and theory of SA. Without repeating too much of the discussion in section 4.3.4, the construction of SA is presented in the form of a pseudocode and some basic concepts of the method are defined. The rest of the chapter is then devoted to the implementation itself and the decisions that were taken during implementation. This involves the discussion of factors that control the performance of the algorithm such as temperature reduction and stopping criteria. Then more problem specific properties are discussed such as how a solution is stored, neighborhood generation, the choice of a starting solution and objective function calculations.

5.1 Simulated Annealing revisited

The ideas that form the basis of SA come from thermodynamics and metallurgy. When a solid crystal is heated to its melting point the atoms can migrate freely in the liquid. When the metal is cooled down, the atoms rearrange in a way that minimizes their potential energy. The final potential energy of the atoms is however strongly dependent on the rate of the cooling. Best results are obtained if thermal equilibrium is allowed to be reached at each temperature level. In order to achieve equilibrium, the rate of cooling must be slow. Rapid cooling tends to freeze the atoms in high energy states which results in sub-optimal crystal structures.

The solution strategy of SA is based on this observation from thermodynamics. For combinatorial optimization problems, the crystal energy can be thought of as different combinations of the integer decision variables. Migration of atoms can be generalized as slight modification to a given solution. The freedom of migration is then controlled by an acceptance probability p which is 1 for those modifications that improve the current solution and is given by $e^{-\Delta/T}$ for all non-improving modifications. For a given temperature T the probability of accepting worse solution depends on the difference in the objective value of the current solution and the new solution $\Delta = f(x_{k+1}) - f(x_k)$. The acceptance probability is then controlled by the control parameter T.

The function $e^{-\Delta/T}$ comes from thermodynamics and is among other things used to calculate the probability that an atom can escape from a potential energy well of depth Δ when the temperature of the system is T. This probability distribution is called the Maxwell Bolzman distribution.

As mentioned in section 4.3.4, the acceptance of non-improving solutions in a controlled manner, is used to explore the solution space by moving in and out of local minimas. Initially almost all solutions are accepted but as the iteration progresses the acceptance probability is decreased by lowering the value of the control parameter T. That has the effect that solutions are chosen more selectively and at the end of the iteration process, only improving solutions are accepted. A pseudocode for the Simulated Annealing algorithm[1] is as follows:

```
Initialize k = 0
Select an initial feasible solution x_0 ∈ S
Select an initial temperature T_0 > 0

while( Stop == true )
    randomly select x_{k+1} ∈ N(x_k)
    Δ = f(x_{k+1}) − f(x_k)
    if( Δ < 0 )
        x_k = x_{k+1}
    else
        generate r ∈ UD(0,1)
        if( r < e^{−Δ/T_k} )
            x_k = x_{k+1}
    T_{k+1} = α(T_k)
    Update stop
    k = k+1
end while
return x_k
```

Here $UD(0,1)$ is the uniform distribution between 0 and 1.

In the beginning of the algorithm an initial solution and starting temperature T_0 are chosen. The initial temperature is usually chosen such that the acceptance probability is close to 1 for all non-improving solutions. This allows the solution space S to be explored properly and diminishes the influence of the starting solution. In each iteration

[1] an algorithm is any well-defined computational procedure that takes some value, or set of values, as an input and produces some value, or set of values as output [23]

a new solution x_{k+1} is generated at random in the neighborhood of the current solution $N(x_k)$. The solution is accepted if it improves the objective value ($\Delta < 0$) or $e^{-\Delta/T}$ is greater than a random number r generated from a uniform distribution between 0 and 1. The temperature is then updated, using the cooling function $\alpha(T)$, and so is the stopping condition. If the solution x_{k+1} is not accepted $x_k = x_{k+1}$ and a new solution is drawn at random in the next iteration.

There are two different types of cooling functions. The temperature can either be reduced continuously or in steps. In the continuous case, the temperature is decreased in each iteration, this is referred to as the **inhomogeneous** version of SA. In the **homogeneous** version of SA the temperature is kept constant for a certain number of iterations before it is reduced. In that case line 13 in the pseudocode is implemented as:

$$\begin{array}{ll} 13a & \textit{if(} k \textit{ mod } L == 0 \textit{)} \\ 13b & \qquad T_{k+1} = \alpha T_k \end{array} \tag{5.1}$$

here $0 < \alpha < 1$, *mod* is the modulus operator and L is the number of iterations for which the temperature is kept constant.

Theoretical studies have shown that the stochastic process generated by the SA algorithm can be modeled using Markov chains. The transition from a solution i to a solution j can be represented in a matrix form, and the probability of moving from solution i to solution j is given by p_{ij}. This probability is only dependent on the two solutions i and j and not on the intermediate steps taken in the transition between them. A constant temperature gives rise to what is called a homogeneous Markov chain, while inhomogeneous Markov chains are obtained when the temperature is not constant. The homogeneous SA can therefore be described theoretically by a sequence of homogeneous Markov chains. By contrast the inhomogeneous version of SA is described by a single inhomogeneous Markov chain.

For the homogeneous version of SA, the Markov chain reaches a stationary distribution after certain number of iterations, if there exists a sequence of steps such that any solution can be transformed to any other solution with non-zero probability. Once a stationary distribution is reached the transition probability p_{ij} becomes independent of temperature. If stationary distribution is reached at each temperature level during the iteration, it has been shown that the form of the stationary distribution tends to a uniform distribution over the set of **optimal** solutions [18]. Hence, the algorithm converges to the optimal solution. In practice the length of the temperature plateau L is too short for a stationary distribution to be reached at each temperature level. Theoretical results have shown that to approximate a stationary distribution sufficiently, the number of iterations at each temperature level needs to be at least quadratic in the size of the solution space [18]. Since the solution space is usually exponential in size this result is only of theoretical interest. It can also be shown that the inhomogeneous version of SA converges asymptotically to the optimal solution. As for the homogeneous case, necessary conditions for convergence can in almost all cases **not** be satisfied in finite time [15].

Even though these convergence results are only of theoretical interests they are useful in two ways:

- They show that the observation of a decreasing potential energy in a cooling metal and its generalization to an optimization heuristic has a theoretical justification. This helps to explain why the method does work for many optimization problems.
- They give guidelines on what factors are important to achieve good results. The results emphasize that the cooling schedule and the reach-ability between different solutions are essential to the performance of algorithm.

The last point emphasises the importance of the neighborhood construction and the cooling schedule chosen in the implementation. During the course of implementation of SA, the decisions that have to be taken can be divided into two groups, *generic-* and *problem-specific*. Generic decisions are those that have to be made for any implementation of SA. They involve the choice of the cooling function $\alpha(T)$, the value of its parameter, initial temperature and stopping criteria. Problem-specific decision are those who are concerned with the neighborhood construction, objective function calculation and other decisions related to the problem being solved. The next two sections discuss these types of decisions made in the implementation of the SA algorithm.

5.2 Generic decisions

Generic decisions basically involve the selection of a cooling schedule. This selection involves finding a value of the initial temperature T_0 and selecting a cooling function and stopping criteria.

When the initial temperature T_0 is selected, the temperature must be high enough to allow almost free exchange between neighboring solutions. Too high values have the disadvantage that the computation time increases unnecessarily. Too low values of T_0 have the effect that the algorithm starts to behave like a descent algorithm, with the risk of converging directly to the nearest local minima in few iterations. The approach used to tune T_0 here was to find two values: T_0^{up} and T_0^{low}, where the upper and lower probability of accepting non-improving moves had the values χ_{low} and χ_{up}. The values of χ were found by estimating the average value of Δ for the first few iterations. Based on this value of Δ, a trial value of T_0 is estimated. The algorithm is then tested with this trial value of T_0 and the accepting rate of non-improving moves calculated. T_0 is then modified and the experiment repeated until the initially selected values for χ are obtained. The values of $T_0 \in [T_0^{Down}, T_0^{up}]$, are then used as a basis when the algorithm is tuned.

In order for the SA algorithm to perform successfully, the cooling schedule must be chosen carefully. The most popular and simple cooling schedule is the *geometric* cooling function:

$$T_{k+1} = \alpha T_k \qquad 0 < \alpha < 1 \tag{5.2}$$

Empirical results have shown that high values of α tend to give best results, $0.8 \leq \alpha \leq 0.99$, with a bias to the higher end of the range [15].

Even though this function can be used for the inhomogeneous version of SA, it is **only** used and discussed here in connection with the homogeneous version of SA. The length of the temperature plateau L in this implementation is constant for each temperature level. Assuming that the temperature is allowed to decrease to 0, only values of the parameters α, T_0 and L need to be decided to determine the cooling schedule. It is worth pointing out that increasing the values of either α or L increases the total number of iterations and results in a slower cooling rate. The geometric cooling schedule was the main cooling schedule used in the implementation and testing of the SA algorithm. The reason for this selection was the simplicity of using this function and it worked well.

A popular cooling scheme used in connection with the inhomogeneous version of SA, is the so called *Lundy-Mees* scheme:

$$T_{k+1} = \frac{T_k}{1 + \beta t_k} \tag{5.3}$$

here β is a suitably small value and can be defined as:

$$\beta = \frac{T_0 - T_f}{M \cdot T_0 \cdot T_f} \tag{5.4}$$

here T_f is the final temperature and M is the total number of neighborhood generations

This cooling scheme was also implemented and tested. This was done to find out if the inhomogeneous version of SA would performer better than the homogeneous version and get a comparison with another type of cooling schedule. The form of the Lundy-Mees and geometric cooling schedule is shown in figure 5.1.

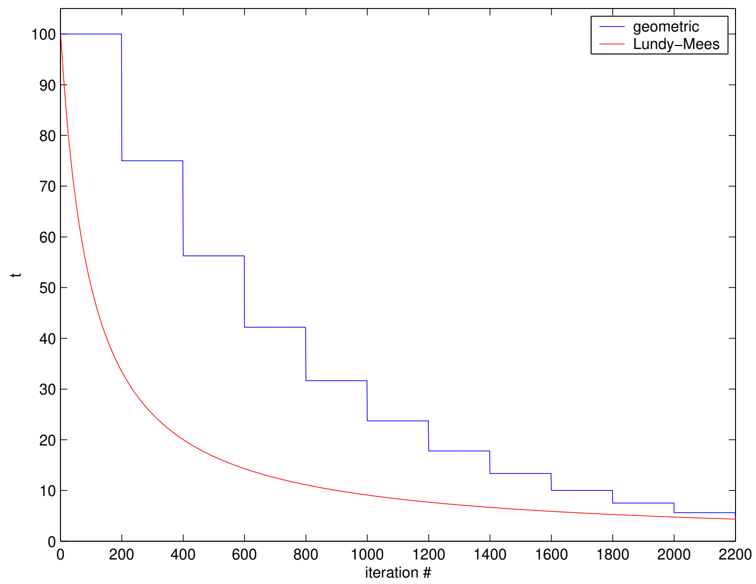

Figure 5.1: The figure shows the form of the Lundy-Mees and geometric cooling schedule.

In theory the stopping criteria for SA is to allow the temperature to decrease to zero. In practice there is however no need to decrease the temperature that far. As the temperature approaches zero the probability of accepting non-improving solutions also approaches zero. One type of a stopping criteria can therefore be some minimum allowed temperature value. When this value is reached, the system is estimated to be "frozen" (there is no possibility of escaping the current local minima). Another method of estimating the "freezing" point is to count the number of iterations since a new solution was accepted.

Two different stopping criterias were used for the SA implemented:

- Stop when the number of iterations since a solution was last accepted reaches a specified limit C_{max}
- Stop when the total number of iterations reaches a specified limit Itr_{max}

The first stopping criteria is used to estimate when the iteration process has been "frozen" in some minima. This parameter needs to be tuned carefully so that it does not compromise the quality of the final solution. The second stopping criteria is simply used as a safety valve if the value of C_{max} has been set too high.

5.3 Problem-specific decision

Problem-specific decisions are mainly concerned with the neighborhood structure, the objective function calculation and the generation of an initial solution. The implementation of the neighborhood structure is crucial to the performance of SA while the implementation of the objective function calculation only effects to computational time. Before discussing these decisions the form of the unit commitment solution is first introduced.

In this implementation each solution is presented in a matrix form where a solution is made up of three different $N \times T$ matrixes describing the unit status (**uc**), electricity production (**ep**) and heat production (**hp**). The time periods ($t = 1, \ldots, T$) are given in the rows of the matrix while different units ($i = 1, \ldots, N$) are given in the columns of the matrix. The form of the unit status matrix is shown in table 5.1.

$$\mathbf{uc} = \begin{bmatrix} u(1,1) & u(1,2) & u(1,3) & \ldots & u(1,T) \\ u(2,1) & u(2,2) & u(2,3) & \ldots & u(2,T) \\ \vdots & \vdots & \vdots & \ddots & \\ u(N,1) & u(N,2) & u(N,3) & \ldots & u(N,T) \end{bmatrix}, \qquad \mathbf{uc}_{i,t} \in \{0,1\}$$

Table 5.1: Form of the solution for the unit schedule.

The subdivision between the urban and rural heat production is made in such a way that the lower part of the matrix belongs to the heat production units in the urban area. The matrixes for electricity and heat production have the same structure as shown in table 5.1 except the production is represented with floating point numbers. During the iteration process it is enough to store only the **uc** matrix and discard the information of the **uc** and **ep** matrixes. The information these matrixes contain are not needed directly during iteration. The value of these matrixes can be obtained in the end by recalculating the EDP using the final **uc** solution.

5.3.1 Neighborhood structure

In the discussion that follows, the word neighborhood structure refers to the selection strategy of choosing a new solution x_{k+1} by modifying the current solution x_k slightly, or $x_{k+1} \in N(x_k)$. A neighborhood algorithm refers to the implementation of this strategy into a finite number of rules or programming statements. There are two different approaches possible for choosing a neighborhood structure:

- **Type I:** $x_{k+1} \in N(x_k)$ is always a feasible solution. The neighborhood algorithm checks the feasibility of each solution generated and discards infeasible solutions.
- **Type II:** $x_{k+1} \in N(x_k)$ can be a feasible or an infeasible solution. The neighborhood algorithm checks the feasibility of each solution generated. If a generated solution is infeasible, a penalty term is included in the objective function to discourage the selection of the solution.

The advantage of using type I neighborhood structure is of course the feasibility of the final solution. Another advantage is that the size of a solution space consisting only of feasible solutions is always smaller than in a solution space where infeasible solutions are included. A disadvantage is that too much computational effort is sometimes needed to generate feasible solutions.

The argument for using a type II neighborhood is that little time is needed for generating each solution. Another advantage is that the reach-ability between two different solutions is generally better compared to type I neighborhood structure. The main disadvantage with type II neighborhood structure is that the final outcome of the SA might be infeasible. In that case, the final solution must be repaired and there is a risk that the resulting solution could be of poor quality.

In the formulation of the UCP used here, the solution space is restricted by the spinning reserve constraint and the minimum up and down constraints. The spinning reserve constraint is relatively easy to handle. If this constraint is not fulfilled, units must be added until enough capacity is reached. The minimum up and down constraints are however difficult to handle. The reason is that it takes considerable computational effort to check (or maintain) feasibility during a solution modification. This is demonstrated in figure 5.2.

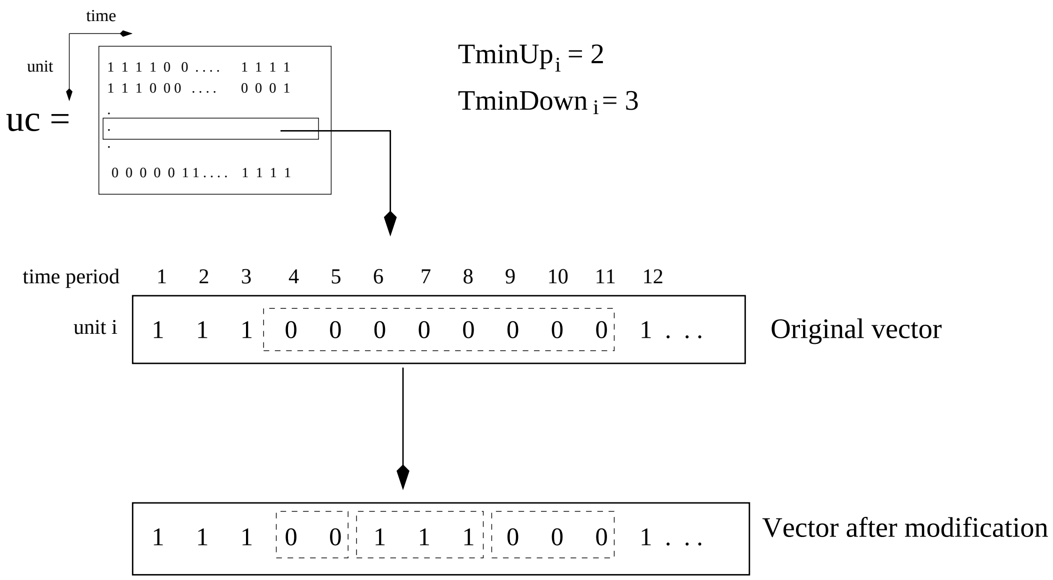

Figure 5.2: The figure shows a modification for the schedule of unit i in time periods 6-8. $TminUp_i$ and $TminDown_i$ are the minimum up and down times of unit i respectively.

The figure shows how unit i is modified by opening the unit in time periods $6-8$. The schedule for unit i is composed of alternating strings of one's and zero's. The neighborhood here consists of all solutions that can be made by changing the operating status of one unit over a few time periods. The modification splits the "zero" string segment in time periods $4-11$ into 3 new segments. After the modification each string needs to be checked for feasibility[2]. The modification clearly results in an infeasible solution in time periods $4-5$. If this solution was a result of type II neighborhood, a penalty would be added to the objective function. However since information is available about the feasibility of each string fragment, it is relatively easy to modify the infeasible solution and change it to a feasible solution (hence type I neighborhood).

The purpose of this example is to demonstrate that there is not much difference in computational effort between the generation of type I and type II neighborhood for this problem. Both types need to generate certain amount of information following a modification and this information can be used to change an infeasible solution into a feasible one. As a result, the minimum up and down constraints are equally difficult to handle for both types of neighborhood. This makes it less attractive to allow infeasible solutions in the neighborhood structure. It must also be taken into account that more iterations are expected to be used for a type II neighborhood since the solution space is less restricted. That means more EDP-calculations which are expensive with respect to computation time.

Concerning the solution reach-ability, it was estimated that a type I neighborhood algorithm did not compromise solution reach-ability if properly implemented. As a result, emphasis was only made on implementing a type I neighborhood algorithm. Two such algorithms called *N1* and *N2* were implemented. The second algorithm is actually an improved version of the first. The two algorithms generate a neighborhood solution by modifying one unit over a certain number of time periods. Other approaches such as changing several units in one time period makes the feasibility of the minimum up and down constraints more difficult to maintain (or verify). Changing only one unit at a time is also more "natural" since the objective is to distinguish between the quality of different units. The rest of this section is devoted to the description of the neighborhood algorithms implemented.

Neighborhood algorithm: N1

This algorithm is the simpler of the two neighborhood algorithms implemented. Its inspiration comes from *A.H. Mantawy et.al.* [10], [13] who used similar approach in the implementation of TS and SA.

The algorithm begins by selecting a unit i and time period t at random. Depending on its value, the element $\mathbf{uc}_{i,t}$ in the unit matrix, either belongs to a string of one's or zero's in row i. The algorithm finds the length of this string l and stores the starting and end position of the string. The modification made to the string dependents on the length l and there are two cases possible

[2]otherwise it would not be possible to reject the solution because of infeasibility for type I neighborhood or add a penalty to the objective function for type II neighborhood.

- **Case 1**. The length l is equal to the minimum up or down time of unit ($TminUp$ and $TminDown$ respectively).
- **Case 2**. The length l is longer than the minimum up or down time of unit i.

In the first case the only possible modification is to change the string completely. This is shown in figure 5.3. Here unit i in time period 6 has been randomly chosen. The length of the string corresponding to $\mathbf{uc}_{i,6}$ is equal to the minimum down time of unit i. The string is therefore changed from 0 to 1 for time periods $4-6$. All other modifications would have resulted in infeasible solutions.

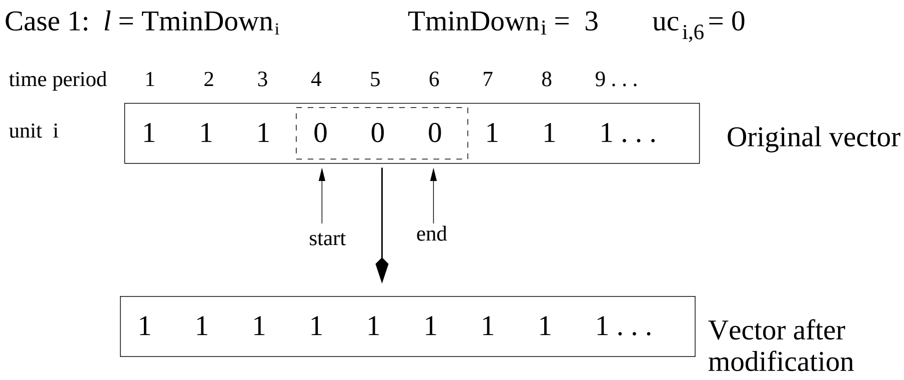

Figure 5.3: Here the string segment is of length 3 which equals the minimum down time of unit i. The only feasible modification is to change the whole string.

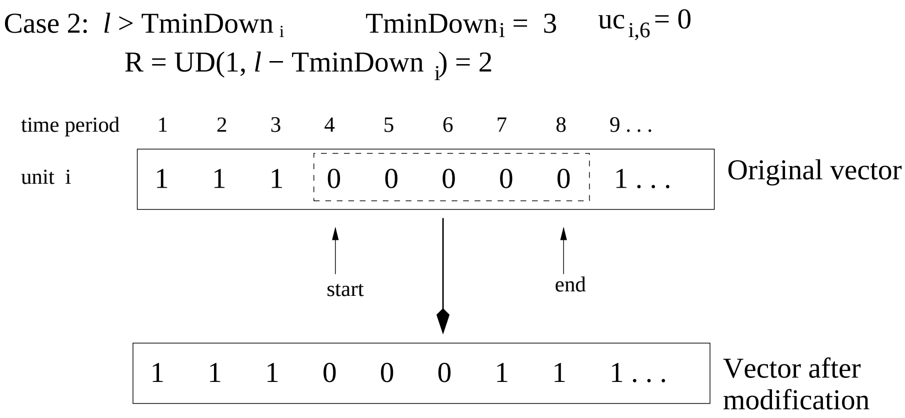

Figure 5.4: Here the string segment is of length 5 which is larger than the minimum down time of unit i. A random number R is generated between 1 and 2 (l-TminDown). Since R = 2, unit i is modified in two time periods, in this case from the end position of the string.

In case 2, the string is modified by generating a random number R between 1 and $l - TminDown_i$ when $\mathbf{uc}_{i,t} = 0$ or $l - TminUp_i$ when $\mathbf{uc}_{i,t} = 1$. The string is then changed for R time periods either in a forward direction starting from the start position of the string or in backward direction starting at the end position of the string. The choice between start or end position is made randomly with equal probability. An example of this case is shown in figure 5.4. In this case the length of the string corresponding to $\mathbf{uc}_{i,6}$ is larger than the minimum down time of unit i. The random number generated equals 2 and the modification in this case is made from the end position.

The pseudocode for the algorithm is as follows:

```
Generate a random unit i ∈ UD(1, N) and time period t ∈ UD(1, T)

if( uc_it == 0 )
    find l = length of the 0-string
    if( l == TminDown_i )
        change the string from 0 to 1
    if( l > TminDown_i )
        generate random number R ∈UD(1, l - TminDown_i )
        choose starting position for modification with probability p = 1/2
        possible positions are either start or end of string
        change the string from 0 to 1 for R time periods
```

The case when $\mathbf{uc}_{it} = 1$ is treated in a similar manner. If the new solution is not feasible with respect to the spinning reserve constraints, the solution is discarded and a new solution must be generated.

This algorithm is fairly simple to implement. The difficulty of the minimum up and down constraints is bypassed by modifying each string segment from the end points only. This leaves behind at least the minimum number of time periods of the original string. The string segments equal to the minimum up and down times are then dealt with separately (case 1).

The disadvantage with this approach is that some solutions have higher probability of selection than others, since modifications are in most cases made from the end points. Modifications to intermediate time periods (middle of a string) have much lower probability of occurring compared to modifications at the end points. An example of this is shown in figure 5.5. In order to open unit i in time periods $6-7$ the string needs to be modified in two steps. The figure shows the shortest possible way that this can bee achieved. In the first modification the random number generated is $R = 7$ and the string is modified from the start position as is needed. If for example R would have been equal to 8 or the string had been modified from the end positions two more steeps would at least be needed to get the desired results. In the second modification R must be equal to 5 for the desired result to be obtained. Even if the first step would be a success the second modification might also lead to some other results which could delay the process of opening units $6-7$. Comparing this case to the possibility of opening unit i in time periods $1-2$ it is obvious that such a modification has much higher probability of occurring.

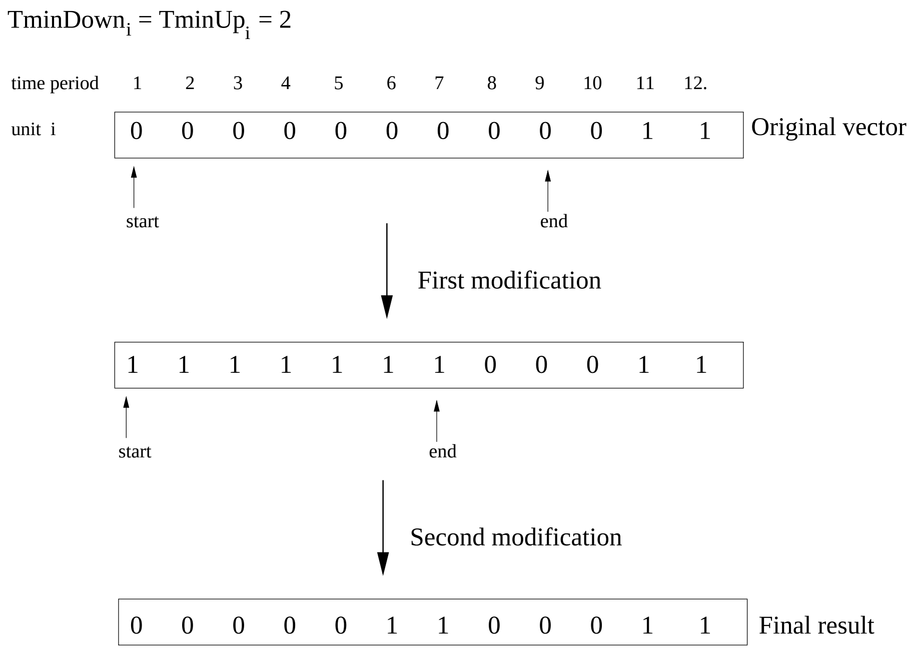

Figure 5.5: The figure shows the shortest possible path to open unit i in time periods $6-7$.

In the implementation used by *A.H. Mantawy et.al* [10], [13] the modification for case 2 was only allowed from the start position of each binary string segment. Compared to that approach, the ability of being able to modify from both ends of a string segment improved the performance of SA both with respect to solution quality and time.

Neighborhood algorithm: N2

This algorithm is an improved version of the first algorithm ($N1$). The bias towards the selection of start and end time periods has now been corrected. Modification can now take place from all positions that can generate feasible solutions. Depending on the length of the binary string l, all points that can be used as a basis for modification are generated and one position is then selected at random. The pseudocode for the algorithm is as follows:

```
Generate a random unit i ∈ UD(1, N) and time period t ∈ UD(1, T)

if( uc_{i,t} == 0 )
        find l = length of the 0-string
         if( l == TminDown_i)
                change the string from 0 to 1
        else
                find the number of feasible time periods
                randomly choose a feasible time period: rT
                if( rT is the beginning or end of string)
                        generate random number R ∈UD(1, l - TminDown_i )
                        change string from 0 to 1 for R time periods staring at start
                        or (end-R+1) depending on which endpoint has been chosen
                 else
                        change string from 0 to 1 for TminDown_i time periods
                        starting at time period rT
```

The case when $\mathbf{uc}_{it} = 1$ is treated in a similar manner. If the new solution is not feasible with respect to the spinning reserve constraints, the solution is discarded and a new solution must be generated.

Depending on the length l, there are 3 possible cases when the number of feasible time periods are calculated. For $u_{i,t} = 0$, these cases can be stated as:

- **Case 1**: $l = TminDown_i$
- **Case 2**: $l > TminDown_i$ and $l < 2 \cdot TminDown_i$
- **Case 3**: $l \geq 2 \cdot TminDown_i$

Case 1 here is identical to case 1 for the $N1$ algorithm. Case 2 here is also the same as case 2 for the $N1$ algorithm except that the length l is now more restricted. For case 3 the feasible positions for modification of the string consists of the *start* and *end* time periods of the string and all time periods t that satisfy the following two conditions:

$$\begin{aligned} &t \geq \textit{start} + TminDown_i \\ &t + TminDown_i \text{ - } \mathit{1+} < \textit{end} \text{ - } TminDown_i \end{aligned} \tag{5.5}$$

The only modification allowed for these time periods is to change the string for $TminDown_i$ number of time periods. The time periods that fulfill these two conditions ensure that there will always be at least $TminDown_i$ number of time periods left from the start and end position of the 0-string segment which is modified. The modification of these positions results in the minimum number of time periods being opened inside the string segment under consideration. Using the example in figure 5.5, this would allow the opening of unit i in time periods $6-7$ directly in one modification. The number of modifications R made for the start and end position in case 3 is the same as for case 2 ($R \in UD(1, l - TminDown_i)$). An example of case 3 is shown in figure 5.6. Since $TminDown_i = 3$ the feasible positions are time periods $4, 7, 8$ and 13. For each time period the following modifications are possible

time period 4: it is possible to change up to 7 time periods starting at position 4
time period 7: change 3 time periods starting at position 7
time period 8: change 3 time periods starting at position 8
time period 13: it is possible to change up to 7 time periods ending at position 13

In this case modification is made from position 7.

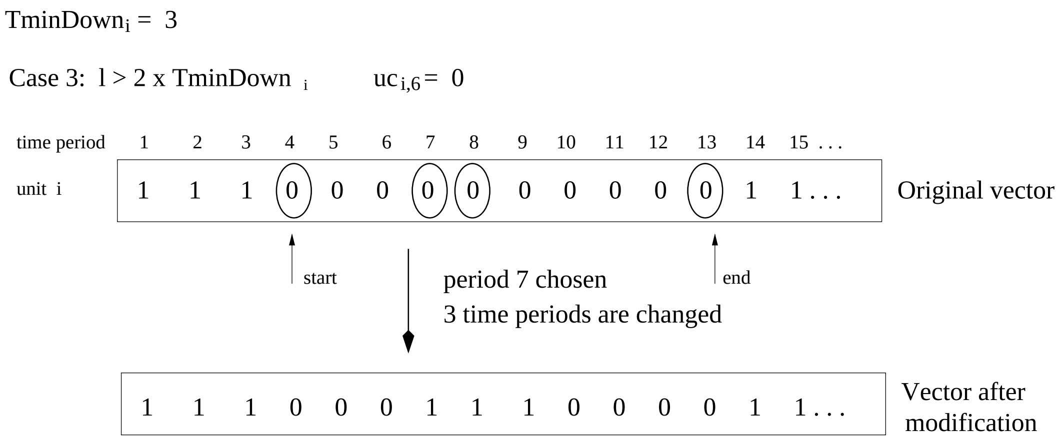

Figure 5.6: The figure shows the feasible start positions (marked with circle) for modifying the string segment in time periods $4-13$. In this case time period 7 is chosen which belongs to case 3. As a result 3 time periods are changed.

The number of modifications that are allowed from the start and end positions in case 3 is quite large. The reason for this choice is that the solution structure was found to consist of long homogeneous schedules for each unit. In many cases, units are open or closed for the whole time horizon. Other units are open for relatively short time. Because of this solution structure, it is desirable to be able to make large modifications from the end points and have the possibility of cutting holes in the intermediate time periods. This modification strategy was found to give the best result with respect to time and objective function value.

5.3.2 Other implementation details

There are two other problem specific decisions that have not been discussed so far but are important to the performance of the SA algorithm. These are the selection of an initial solution for the algorithm and the objective function calculation. The implementation of both these decisions will now be discussed shortly.

Objective function calculation

In each iteration of the SA algorithm, the difference between the current solution x_k and the newly generated solution x_{k+1} is that unit i has been modified over a time horizon of R time periods. Instead of recalculating the whole objective value, which means that the EDP is solved for all time periods in the unit matrix **uc**, it is enough to simply update the objective function. This is possible because the time periods are independent in the economic dispatch problem[3]. Instead of solving the EDP for the whole unit matrix, only those time periods that are changed need to be recalculated. It is also necessary to check after each modification if the number of start-up periods for unit i has increased or decreased. The updating of the objective function is then given by:

$$f(x_{k+1}) = f(x_k) + \Big(\sum_{t=start}^{end} EDP_{k+1}(t) - EDP_k(t) \Big) + s_c(i) \cdot \Big(Num^s_{k+1}(i) - Num^s_k(i) \Big) \tag{5.6}$$

Here

- $EDP_k(t)$ is the production cost from the economic dispatch problem in time period t for the solution in iteration k.
- $Num^s_k(i)$ is the number times unit i is started for the solution in iteration k.
- $s_c(i)$ is the start-up cost of unit i.

By updating the objective value in this way the computation time is decreased by 70% compared to not updating the objective value.

Generating an initial solution

Because of the random nature of SA, it is desirable to use different initial solutions to estimate the average performance of the algorithm. The initial solution can have effect on the algorithms performance, but the effect usually decreases if a sufficient number of iterations are made at high temperatures. The generation of an initial feasible solution was made by using the neighborhood algorithm $N1$. An algorithm was implemented to generate initial feasible solutions. Solutions are generated as follows:

The algorithm initiates all elements in the unit matrix **uc** to 1. This corresponds to the maximum capacity of the power production system. The neighborhood algorithm $N1$ is then repeatedly used to turn units on or off for a certain number of iterations.

[3] remember that time periods are only dependent because of the start-up cost. In the EDP the values of $u(i,t)$ are fixed and the start up cost is simply a constant

The probability that a unit is turned on can be controlled. This is done by using an acceptance probability p which determines if a unit should be turned on or not. There is however no restriction on turning units off. This gives the possibility of controlling the capacity of the initial solution. Units are however not selected based on their production cost so the initial solutions are usually quite far from any local optimum. A pseudocode for the algorithm is as follows:

Choose an acceptance probability $p \in (0,1)$
Initiate $\mathbf{uc}_{i,t} = 1 \quad \forall i \in N$ *and* $\forall t \in T$

    while($k < MAX$ *)*
        Generate a random unit $i \in UD(1,N)$ *and time period* $t \in UD(1,T)$
        if($\mathbf{uc}_{i,t} == 1$ *)*
            execute neighborhood algorithm N1 for the case $\mathbf{uc}_{i,t} = 1$
        else
            generate a random number $r \in UD(0,1)$
            if($r < p$ *)*
            execute neighborhood algorithm N1 for the case $\mathbf{uc}_{i,t} = 0$
        $k = k + 1$
    end while

5.4 Conclusion

The subject of this chapter has been the theory and implementation of Simulating Annealing. Various implementation details were addressed such as the solution form, objective function calculations, stopping criteria and generation of initial solution. The main emphasis was however made on describing the cooling schedule and the neighborhood algorithms that were used. The reason being that they are crucial to the performance of SA.

The selection of cooling schedule made in this chapter is quite traditional. The geometric cooling function and the Lundy-Mees function are frequently used as first choice for the homogeneous and inhomogeneous version of SA respectively. The reason why both schedules were implemented was to check if either the homogeneous or inhomogeneous version of SA worked better. The geometric cooling schedule was given more weight in the experimentation since it turned out to be both simple to work with and robust. Alternative cooling schedules did not have to be tested since satisfactory results were obtained by the current choice.

The two neighborhood algorithms that were implemented both maintain feasibility and use similar approach for generating solutions. Even though they are similar, there is one crucial difference between them. The $N2$ algorithm is significantly more difficult to implement properly and gives more flexibility in generating new solutions. The reason for including the $N1$ algorithm in the discussion is that simplicity with respect to implementation is always a desired quality when a selection is made. It is therefore interesting to compare how such a simple approach performs compared to a more sophisticated approach. The approach used by *A.H. Mantawy et.al.* [10], [13] that was the inspiration for the construction of the $N1$ and $N2$ algorithms, performed significantly worse and was therefore omitted from discussion.

It would have been interesting to implement a type II neighborhood algorithm and compare to the two type I algorithms implemented. Unfortunately, a lack of time prevented that.

Chapter 6

Experimental results

This chapter finally presents the results for the solution approach of using Simulated Annealing and the EDP heuristic from chapter 3 to solve the unit commitment problem formulated in chapter 2. Before presenting the experimental results, a short overview of the actual problem is presented. The results are then divided into three main parts:

The first part focuses on the tuning and testing of the Simulated Annealing algorithm. This involves comparing the two different neighborhood algorithms and the selection of a cooling schedule. The effect of the initial starting solution is also investigated. Finally, experiments that were made with some possible modifications to the basic SA are presented. The purpose of these experiments is to show how the SA algorithm responds to different generic and problem specific decisions for this problem. The results from these experiments are then used to select a neighborhood algorithm and a cooling schedule which is used as a basis in later tests.

Given these test results, the second part of this chapter is devoted to the performance of Simulated Annealing on the problem itself. Throughout the experiments, the results from SA are compared to optimal or lower bound values obtained by GAMS[1]. In addition to comparing the solutions of the SA algorithm to optimal values, considerable emphasis is made on testing different problem parameters and how they effect the performance of SA. The problem parameters that are tested are: *start-up cost*, *minimum unit capacity* and *minimum up and down times*. All three parameters had to be estimated in the problem formulation. In addition to these tests, the effect of the time horizon and the number of units is also investigated.

The chapter is then concluded by giving a short comparison of the performance of SA algorithm implemented in this report and the Tabu Search implemented by a fellow student working on the same project.

[1]As mentioned before, GAMS uses CPLEX as a solver which has a Branch & Bound routine

6.1 Problem overview

This section gives a short representation of the power production system under consideration. This is done to give the reader some idea about the actual problem without referring only to appendix A were the whole data for this problem is given.

The power production system consists of 25 different units where 10 units belong to the urban area and 15 to the rural area. Table 6.1 shows the number of units for each type in the urban area. The table also gives the range of the maximum and minimum capacity of these units. The capacity values correspond to electricity capacity for all unit types except heat only boilers. In table 6.1 the minimum capacity range for extraction units means that for the three units, the minimum production capacity is between 109-268MW for electricity production. Table 6.2 gives the same information for the rural area:

	Urban area		
type	number	min capacity range	max capacity range
Extraction	3	109-268MW	272-538MW
Back pressure	4	9-92MW	35-230MW
Condensing	0	xxx	xxx
Heat boiler	3	75-100MW	400-1500MW

Table 6.1: Number of units for each type in the urban area and their capacity range.

	Rural area		
type	number	min capacity range	max capacity range
Extraction	1	320MW	640MW
Back pressure	6	8-97MW	30-243MW
Condensing	4	78-414MW	196-828MW
Heat boiler	4	8-15MW	30-60MW

Table 6.2: Number of units for each type in the rural area and their capacity range.

In back pressure units the maximum heat capacity is 2-3 times larger than electricity capacity but in extraction units heat capacity is around 10% larger. The total production capacity of the production system is:

- Electricity capacity: 4851MW
- Urban heat capacity: 6131MW
- Rural heat capacity: 1870MW

The maximum electricity demand is around 2300MW while the maximum heat demand is around 1500MW and 1200MW for urban and rural area respectively. The large heat capacity in the urban area is a little bit strange but the reason for this capacity are two 1500MW heat only boilers.

The minimum up and down times are 2-5 hours depending on unit type and capacity. Condensing units with high production cost and heat only boilers are considered as peak units and can be started and shut down in only 2 hours.

The start-up cost is estimated to be the same as the production cost at maximum capacity. This does not apply for peak units where the start-up cost is kept lower. The spinning reserve constant used throughout the experiments is 8% of the demand in each time period for both electricity and heat production.

6.2 Testing and tuning SA

In this section the SA algorithm is tested on the two different neighborhood algorithms and cooling functions presented in chapter 5. The better neighborhood algorithm and cooling function is then selected and used for the remaining tests. The effect of different initial solutions and some technical improvements are also examined. The reason for reporting on these tests is to demonstrate which generic and problem specific decisions are important to the performance of the SA algorithm for this problem. Unless otherwise specified the time horizon used in the tests is **24** hours. In many cases a box plot is used to present the test results. The reason for this choice is that box plots make it is easy to visualize the variance within and between different tests. The box plot used here has the following properties:

- The lower and upper lines of the "box" are the 25th and 75th percentiles of the sample. The distance between the top and bottom of the box is the interquartile range.
- The line in the middle of the box is the sample median. If the median is not centered in the box, that is an indication of skewness.
- The "whiskers" are lines extending above and below the box. They show the extent of the rest of the sample.
- Outliers are data with values beyond the ends of the whiskers. An outlier is a value that is more than 1.5 times the interquartile range away from the top or bottom of the box. An outlier is marked with a red '+'.

6.2.1 Selecting a cooling schedule

To determine a cooling schedule the geometric cooling function (equation 5.1) was first tuned using the neighborhood algorithm $N2$. The length of the each temperature plateau

L, is the same at all temperatures. After some initial experimentation the following values were used as a basis for selecting a cooling schedule:

$\alpha = \{0.99, 0.95, 0.90\}$
$L = \{500, 400, 300\}$
$T_0 = \{10000, 8000, 5000\}$

These initial temperature values were found to give the following acceptance probability for non-improving solutions:

$\chi \approx \{0.90, 0.80, 0.70\}$

The stopping criteria used in these tests was to iterate until 400 iterations had passed since a solution was last accepted ($C_{max} = 400$). This value is far too high but it was adjusted after the best value for the reduction parameter α, the initial temperature T_0 and the temperature plateau L had been selected. Each test was repeated 20 times using a randomly generated initial solution in each run. Table 6.3 shows in what order the parameters are permuted and the test results are shown in figure 6.1.

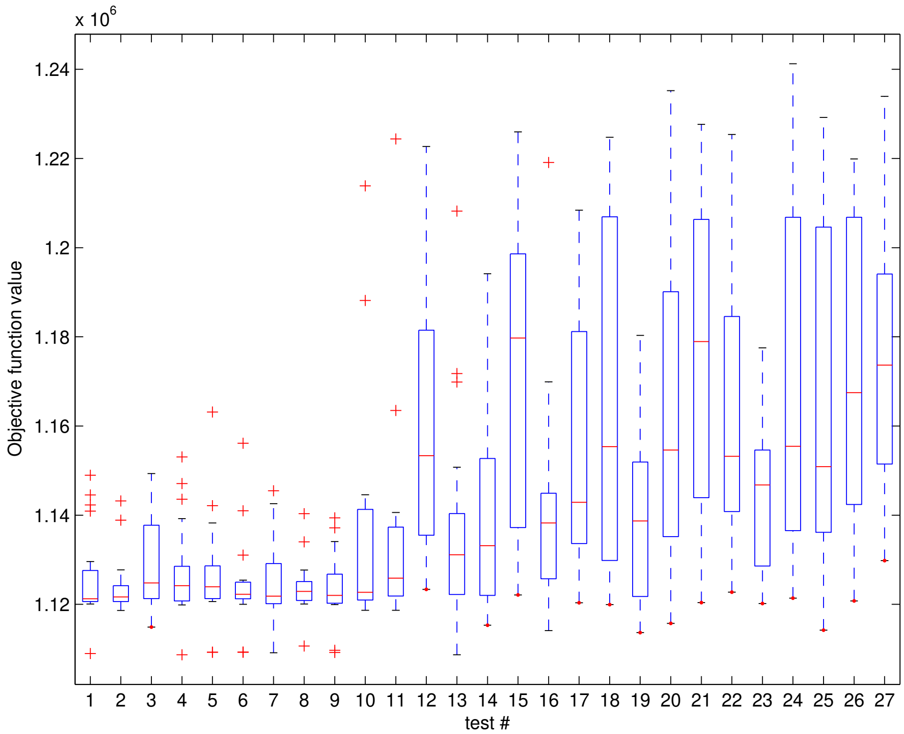

Figure 6.1: The figure shows the result of tuning the parameters α, L and T_0 using the geometric function. Each box is composed of 20 different runs.

The box plot clearly shows that the median and the spread in the data increases as the cooling rate becomes faster. There is a significant change in solution quality from

test ♯	1	2	3	4	5	6	7	8	9	10 ...	27
α	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.95 ...	0.90
T_0	10000	8000	5000	10000	8000	5000	10000	8000	5000	10000 ...	5000
L	500	500	500	400	400	400	300	300	300	500 ...	300

Table 6.3: Permutation of parameters used for tuning SA.

test 10 were the rate of the cooling is increased by the reduction of α from 0.99 to 0.95. Tests 1,2 and 4-9 give similar results with respect to objective value and spread in the data. Test number 1 has a computation time of 27 seconds but the computation time decreases continuously down to approximately 2 seconds for test 27. Based on the objective value, computation time and small spread in the data, parameters from test ♯8 were selected. The computation time for this selection of parameters is 19 seconds. The value of C_{max} was then tuned separately and could be reduced down to $C_{max} = 300$ without affecting the solution quality. This only lead to about 2 seconds reduction in computation time.

The parameter tuning for the Lundy-Mees function (equation 5.3) was made in a similar way as for the geometric cooling function. In this case, it was only necessary to vary the parameters T_0 and β. This function was found to be much more sensitive with respect to parameter values than the geometric function. The results were also worse than those obtained by the geometric function in all cases. The best result found with the geometric function and the Lundy-Mees function is shown in figure 6.2. There is a significant difference in mean value between these two tests and that was verified using a t-test at 5% significance level. The variance in these two tests is however almost the same. This difference in mean value was one reason why the Lund-Mees function was excluded from further experiments. Another reason was that it performed much worse than the geometric function with respect to computation time. In the test in figure 6.2 the computation time is 95 seconds for the Lund-Mees function compared to 17 seconds for the geometric function. The geometric function is the only cooling schedule used for the tests presented in the remainder of this chapter.

The test strategy that has been presented here was used throughout the experiments in this chapter. If some changes were made such as increasing the time horizon , changing neighborhood algorithm or modifying the problem data, the cooling schedule was retuned using this approach.

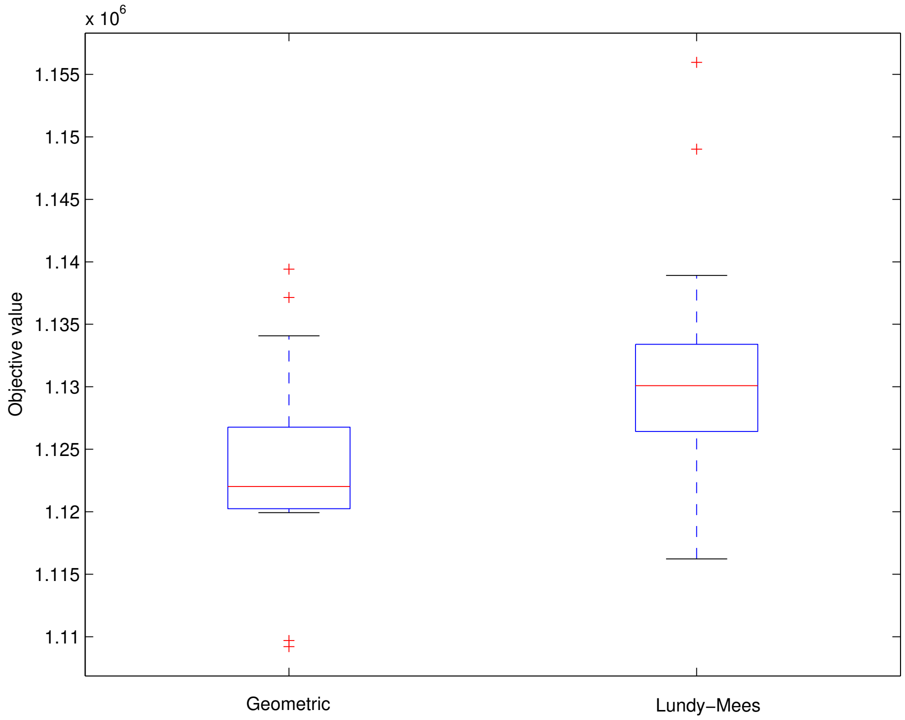

Figure 6.2: The figure shows the difference between the best results obtained by the geometric function and the Lundy-Mees function. Each box is composed of 20 different runs.

6.2.2 Testing different neighborhoods

Both neighborhood algorithms ($N1$ and $N2$) were tested for a time horizon ranging from 1-4 days[2]. The results for these tests are shown in figure 6.3. Each test is an average of 20 runs and the mean μ and variance σ are plotted: $\mu \pm \sigma$. The cooling schedule is tuned separately for each time horizon and neighborhood algorithm. In order to get a better visual comparison, the mean for each test in figure 6.3 has been divided by the number of days in the test. The objective values differ between tests because the demand is generated with random variance between days. The percent difference between the two algorithms in mean value and variance is shown in table 6.4.

	Day 1	*Day 2*	*Day 3*	*Day 4*
difference in μ	1.0%	2.2%	2.4%	2.4%
difference in σ	49%	57%	42%	27%

Table 6.4: Results from testing the two neighborhood algorithms.

The neighborhood algorithm $N2$ performs better than $N1$ in all cases, both with respect to variance and mean value. For a time horizon between 1 and 3 days, the variance for $N1$ is 50% larger than the variance for $N2$ but this difference is reduced to 27% when the time horizon is 4 days. The conclusion that can be drawn from this test is that the approach of allowing modifications only from the end of each string segment as the $N1$

[2] 24-96 hours

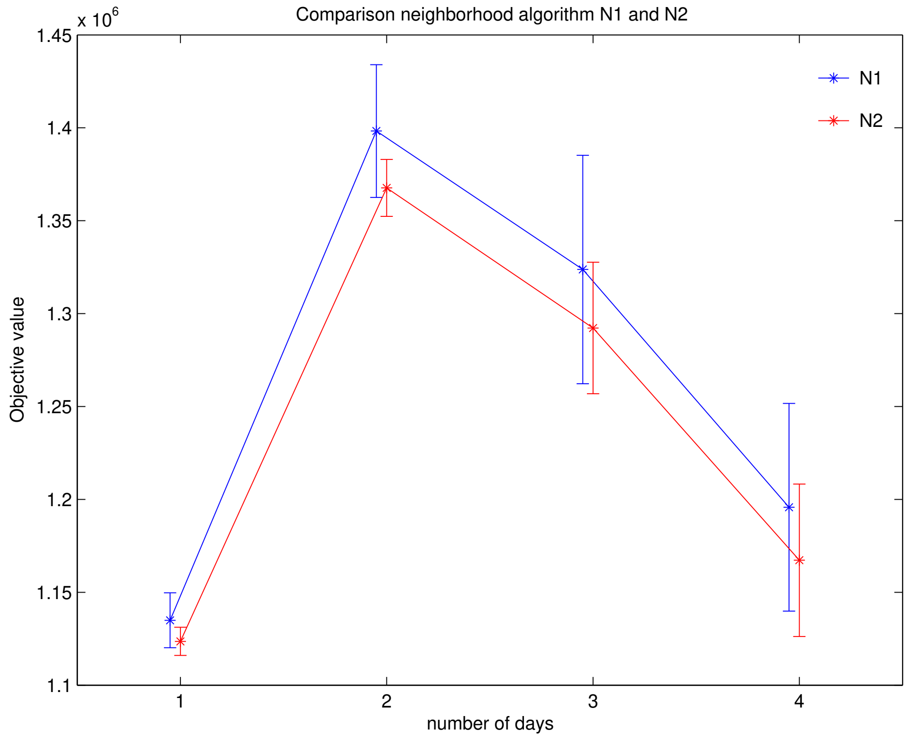

Figure 6.3: The figure shows the difference between the two neighborhood algorithms $N1$ and $N2$. For each string segment $N1$ allows modifications from endpoints only but $N2$ allows modification from all feasible points on the string segment. The performance of $N2$ is significantly better in all cases.

algorithm does, gives higher variance and mean compared to allowing modifications to take place from all feasible positions of each string segment as the $N2$ algorithm does. The SA algorithm converges on the other hand faster when the $N1$ algorithm is used. The difference in computation time is 25% and increases continuously up to 70% for a time horizon of four days. This difference in computation time is probably due to the fact that the neighborhood size is much larger for the $N2$ algorithm. The $N2$ algorithm allows modifications to take place at all feasible positions in a given string segment (see section 5.3.1) but $N1$ only allows modifications at the end points. Even though the $N1$ algorithm performed better with respect to time it gives worse solutions on average and shows higher variance than $N2$, which makes it less robust. For that reason the $N1$ algorithm was excluded from further use and the results reported in the remainder of this chapter are made with $N2$.

6.2.3 Testing different initial solutions

After having selected a neighborhood algorithm and a cooling schedule, the next logical step is to find out what effect the initial solution has on the performance of the SA algorithm. The initial solution algorithm discussed in section 5.3.2 can control the production capacity of the solution it generates. The effect of the initial capacity on the performance of SA was investigated and the result was that the final solution was independent of the initial capacity of the system. This is perhaps not so strange since the production capacity of a given solution is by no means an estimator of solution quality with respect to the objective function.

Another test, which took the objective value into account, was therefore performed. This was done by sampling 5 different solutions from a trajectory of a descent algorithm. Each of these solutions were then used as an initial solution for SA. Each experiment was repeated 20 times and the result is shown in figure 6.4.

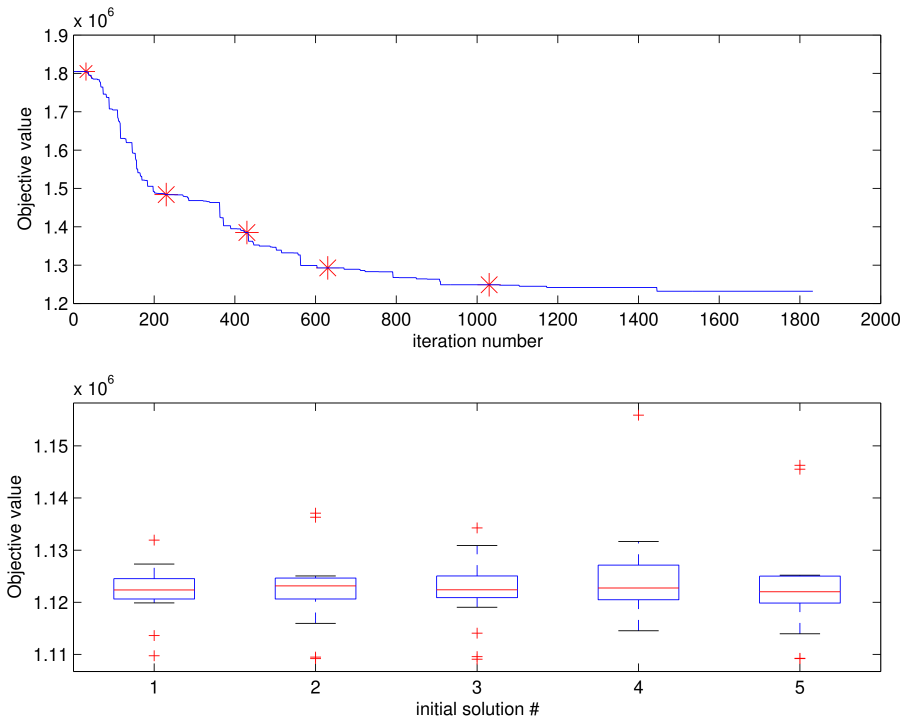

Figure 6.4: The upper figure shows the iteration trajectory from a descent algorithm. The points marked with a red asterisk (*) are sampled and used as an initial starting point for SA. The final result of the SA iteration for each point is shown in the lower figure.

The figure clearly shows that there is no significant difference between these five solutions. A t-test also confirmed that this was the case. The spread of the final solution in each test is also of the same magnitude as is observed when a random initial solution is generated in each run (compare for example with figure 6.2). It is therefore concluded that the SA algorithm is in this case independent of the initial solution, at least for a time horizon of 24 hours. For the tests made in the remainder of this chapter, each run of the SA uses a random starting point which is generated by the algorithm described in section 5.3.2.

6.2.4 Possible improvements

At this stage a suitable cooling schedule and a neighborhood algorithm have been selected for use. It has also been shown that the algorithm is not sensitive to the initial solution, at least for a time horizon of 24 hours. To conclude the testing phase of the algorithm it was decided to experiment with three different modifications to the basic SA algorithm, which have been suggested in the literature [15], [18]. These modifications are:

1. **Locally optimized SA**
2. **Storing the best solution found**
3. **Approximating the exponential function**

Locally optimized SA

In locally optimized SA a new solution x_{k+1} is generated by sampling a certain number of solutions in the neighborhood of the current solution and selecting the best one. This can in some cases lead to better final solutions since each neighborhood is explored better than when only one solution is generated at random.

In this case, locally optimized SA did not result in a better performance compared to basic SA. The quality of the final solution became worse as the number of solutions sampled in each neighborhood increased. When the number of sampled solutions was larger than 30 the algorithm worked effectively as a descent algorithm. Figure 6.5 shows the iteration trajectory of a descent algorithm, locally optimized SA and basic SA. All three runs use the same random seed[3] and the number of points sampled for both the descent algorithm and the locally optimized SA in each iteration is 10 solutions. Because the descent algorithm has converged after relatively few iterations the final solution is extended for comparison.

The figure shows that the solutions for the locally optimized SA decrease much faster than for basic SA initially, but it levels off much sooner than for basic SA. The final result is midway between the solution generated with basic SA and the descent algorithm. The reason for this poor performance is that the locally optimized SA has much higher probability of generating good solution in each iteration than basic SA and that prohibits an escape from local minimas. For locally optimized SA, each non-improving solution accepted has a high probability of being compensated for quickly with a good solution and the algorithm starts to cycle back and forth in the same local minima.

[3]the random number generator generates the same series of random numbers for all three tests.

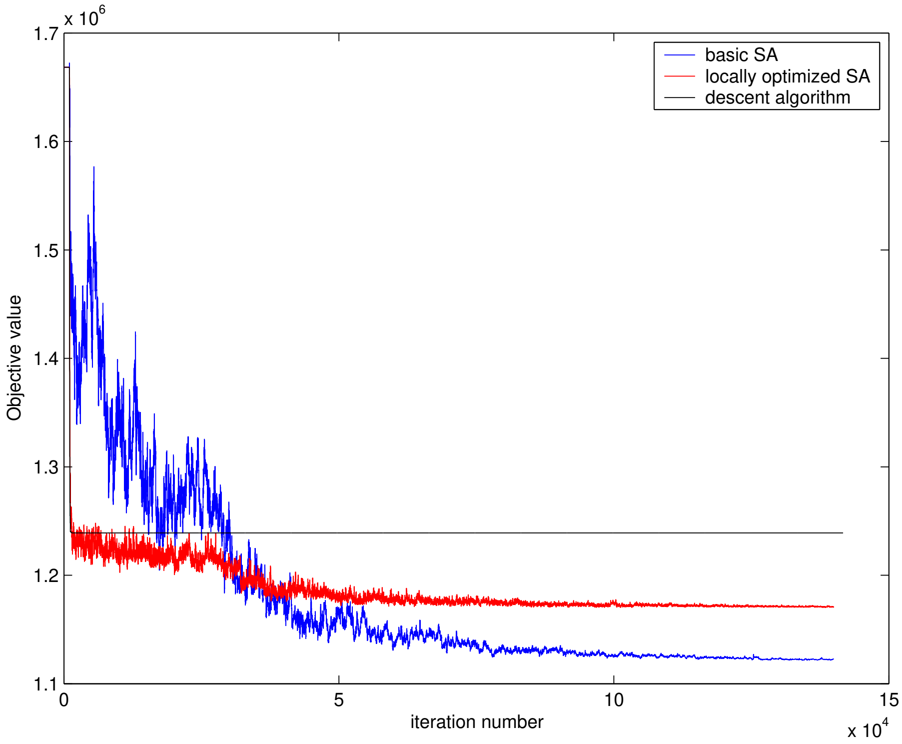

Figure 6.5: The figure shows the iteration trajectory for basic Simulated Annealing, a descent algorithm and locally optimized Simulated Annealing. The iteration trajectory for the descent algorithm has been extended to give comparison to the other two methods.

By comparing the iteration trajectory of basic SA and the descent algorithm figure 6.5 shows that the improvement in objective value that is made by using basic SA is approximately 10%. The price for this 10% improvement is approximately 120×10^3 iterations.

Storing the best objective value

The argument for storing the best objective value found so far during the iteration process is that there is some probability that the algorithm moves out of a "good" local optimum somewhere along the way. For neighborhoods where solution reach-ability is poor it can happen that a non-improving solution, which is difficult to escape from, is accepted. The final solution could therefore be much worse than what has already been found before.

The result from this experiment was that storing the best objective value did not give better results. This was tested repeatedly but the only effect observed was a 40% increase in computation time. The reason for the increase in computation time is that the unit matrix needs to be copied frequently when the best solution is updated. That is a computationally expensive operation. This increase in computation time can be partly avoided by only storing the best values at low temperatures (at high temperatures most solutions are accepted anyway). Since this modification did not lead to improvements it was omitted.

Approximating the exponential function

The implementation of the SA algorithm was written in C++ which calculates the exponential function by using Taylor series. Each estimation is quite expensive with respect to time since many floating point operations are needed. One way to reduce this overhead is to approximate the exponential $e^{\frac{-\Delta}{T}}$ by only using the first term in the Taylor expansion $1 - \frac{\Delta}{T}$. Using this approximation resulted in 15% reduction in computation time on average. No difference was detected in quality of the final solution. Unfortunately, many tests had already been completed when this was discovered and it was therefore decided to continue using the exponential function instead of this approximation.

6.3 The performance of SA

In the last section emphasis was put on experimenting with different parameters and attributes that are common with all implementations of SA. Having done that, the next logical step is to measure the performance of the SA algorithm on the problem itself. The performance of the SA algorithm is estimated by comparing its solutions to optimal values which are calculated in GAMS. In most cases GAMS is able to find optimal solutions to the problem but when that is not possible, GAMS provides a lower bound which is used.

When these performance tests are made, it is important to keep in mind that the underlying data is only an estimate and does include errors. It is therefore likely that the data might at some stage have to be modified and for that reason, it is important to know how the SA algorithm responds to these changes. The problem parameters that are relevant to the performance of SA in this case are the minimum up and down times, the start-up cost and the minimum production capacity. During the problem formulation, the values for these parameters had to be guessed since they were not included in the *Balmorel* model. It is likely that some of the choices made when these parameters were estimated could be better adjusted. The first half of this section is therefore devoted to the experimentation of these parameters.

In the second half of this section the effect of the time horizon is investigated. This section is then concluded by looking at the effect of increasing the number of units in the system. Throughout this section an example of the solution structure is given when it is considered relevant to the discussion.

6.3.1 The effect of start-up cost

The start-up cost used in this formulation is estimated as the production cost at maximum capacity, except for peak units where it is kept lower. This value is only about 1-2% of the objective value but has a significant effect on the solution structure. The effect of the start-up cost on the performance of SA was investigated by using six different variations of the original value as shown in table 6.5

multiple of original	test 1	test 2	test 3	test 4	test 5	test 6	test 7
start-up cost	8	4	3	2	1	0.5	0

Table 6.5: Shows the order in which different start-up cost were tested.

Test number 5 uses the original start-up cost and in test number 7 the start-up cost is omitted. The time horizon used in these tests is 24 hours and each test is composed of 20 different runs of the SA algorithm. The optimal value for each test is calculated in GAMS for comparison. The results are shown in figure 6.6. The results from each test are represented by a box plot and the optimal value is plotted below (marked with *). The mean value μ, variance σ and percent deviation of μ from the optimal value for each test are is shown in table 6.6

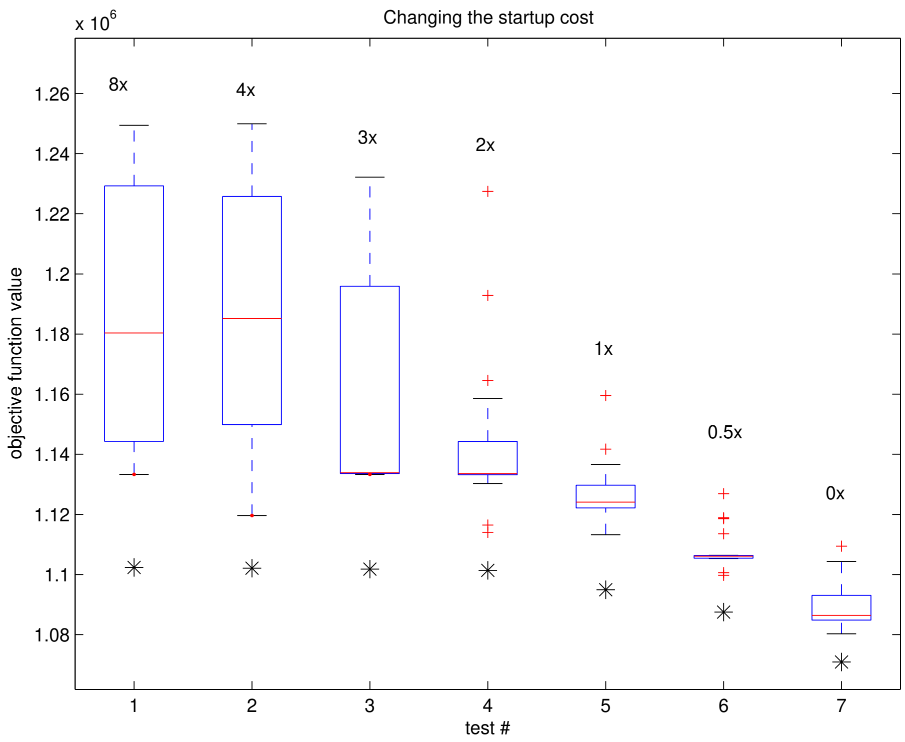

Figure 6.6: The figure shows the performance of the SA algorithm for different multiplies of the original start-up cost(8x, 4x, 3x, 2x, 1x, 0.5x and 0x). Each test composed of 20 different runs. The optimal value for each test is marked with an asterisk (*).

test ♯	μ	σ	% error
1	1.1880×10^6	0.043×10^6	7.8
2	1.1865×10^6	0.042×10^6	7.7
3	1.1603×10^6	0.040×10^6	5.3
4	1.1430×10^6	0.026×10^6	3.8
5	1.1270×10^6	0.010×10^6	2.9
6	1.1080×10^6	0.006×10^6	1.9
7	1.0896×10^6	0.007×10^6	1.7

Table 6.6: Shows mean variance and % deviation from the optimal solution for the 7 tests.

What is interesting about these results, is that both the variance and deviation from optimal value increase significantly when the start-up cost is increased. This increase levels off in test ♯3 where the original start-up cost has been multiplied by 4. Decreasing the start-up cost results in slightly better performance and decrease in variance. In all cases the deviation from the optimal value is less than 10% on average, which is quite acceptable for a heuristic method. For the original start-up cost the deviation is only 3% on average which is quite satisfactory.

It is also worth noting that the optimal value changes very little once the start-up cost has been doubled (test ♯4). The reason for this behavior is that an increase in start-up cost is compensated for by decreasing the number of start-ups. A possible explanation of the larger deviation from optimum with increased start-up cost can to some extent be explained by looking at the solution structure of the final solution for these tests. The solution structure for the tests 4, 5 and 6 are shown in figures 6.7-6.12, both for the optimal value and the SA solution. The solution structure is represented in two different plots:

- **Capacity plot:** This plot is composed of three different figures. The top figure shows the electricity production, the middle figure gives the urban heat production and the bottom figure gives the rural heat production. In each figure the maximum and minimum capacity of the system is represented with a red line. The demand is given by a blue line and the total production in each time period is marked with an asterisk (*). The optimal solution is given to the left on these plots while the SA solution is given on the right.

- **Image plot:** This plot is a graphical representation of the unit matrix **uc**. The matrix which is composed 1's and 0's is displayed as an image object. Black means that a unit is active and white means it is closed. The time horizon is given on the x-axis and units are marked on the y-axis. The optimal solution is given to the left on these plots while the SA solution is given on the right.

The solution for the original start-up cost is given first in figures 6.7 and 6.8. After that the solution for the reduction in start-up cost by half (test ♯6) is given in figures 6.9 and 6.10. Finally the results for the case were the start-up cost has been doubled (test ♯4) are given in figures 6.11 and 6.12.

Original start-up cost

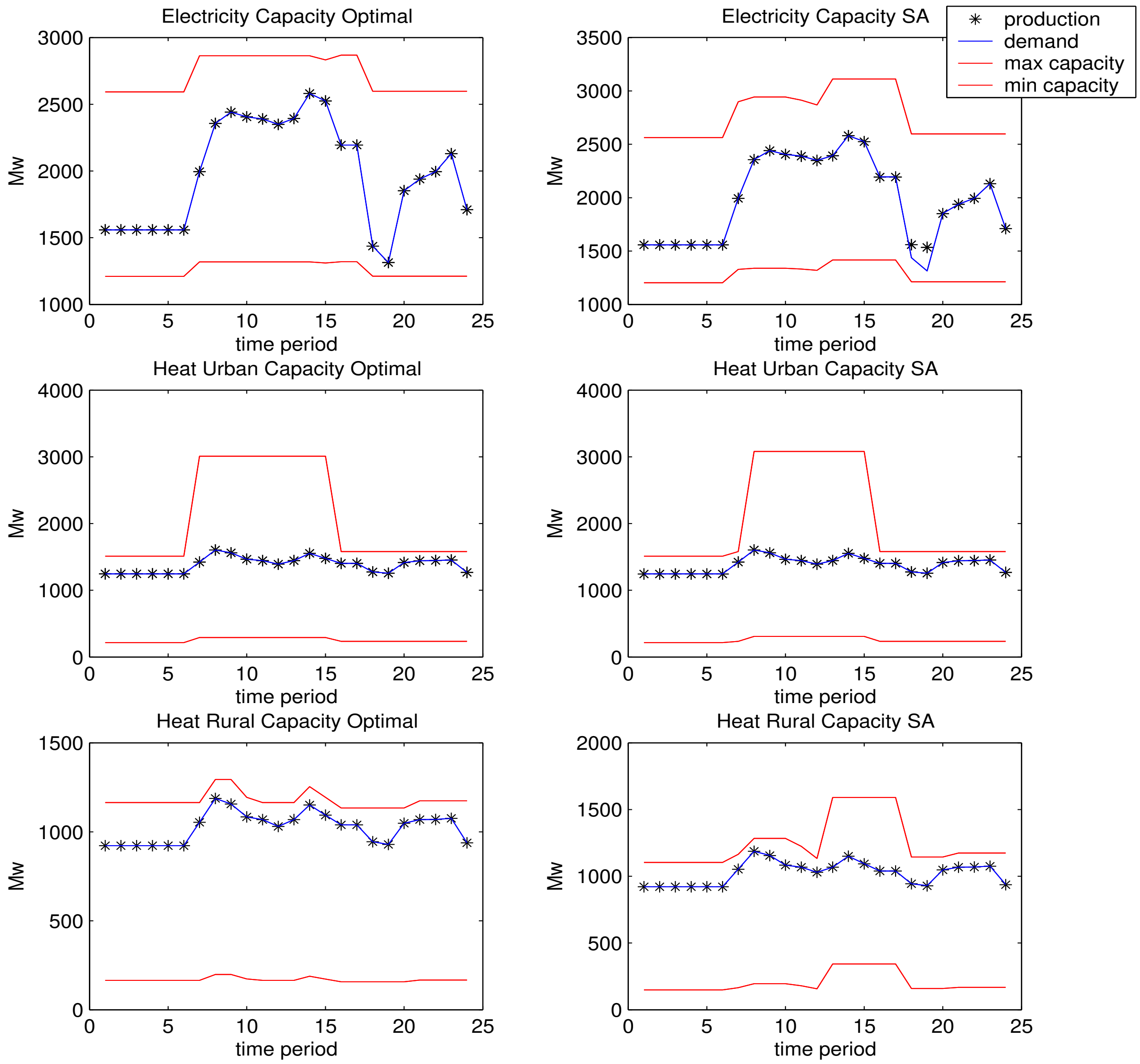

Figure 6.7: Capacity plot for a solution from test ♯5. The start-up cost is unchanged in this case.

Looking at the top figure in figure 6.7, reveals that the increase in electricity demand in time periods 6-17 is compensated for by opening of extra units. When the demand decreases sharply in periods 18 and 19 the same units are closed down. For the urban heat production, an increase in heat demand is compensated for by opening one 1500MW heat boiler, which is then closed down once the peak periods are over. For the rural heat production, there is much more dynamic in opening and closing of units. The main reason for this is that the rural area has smaller units and some of them have low start-up cost.

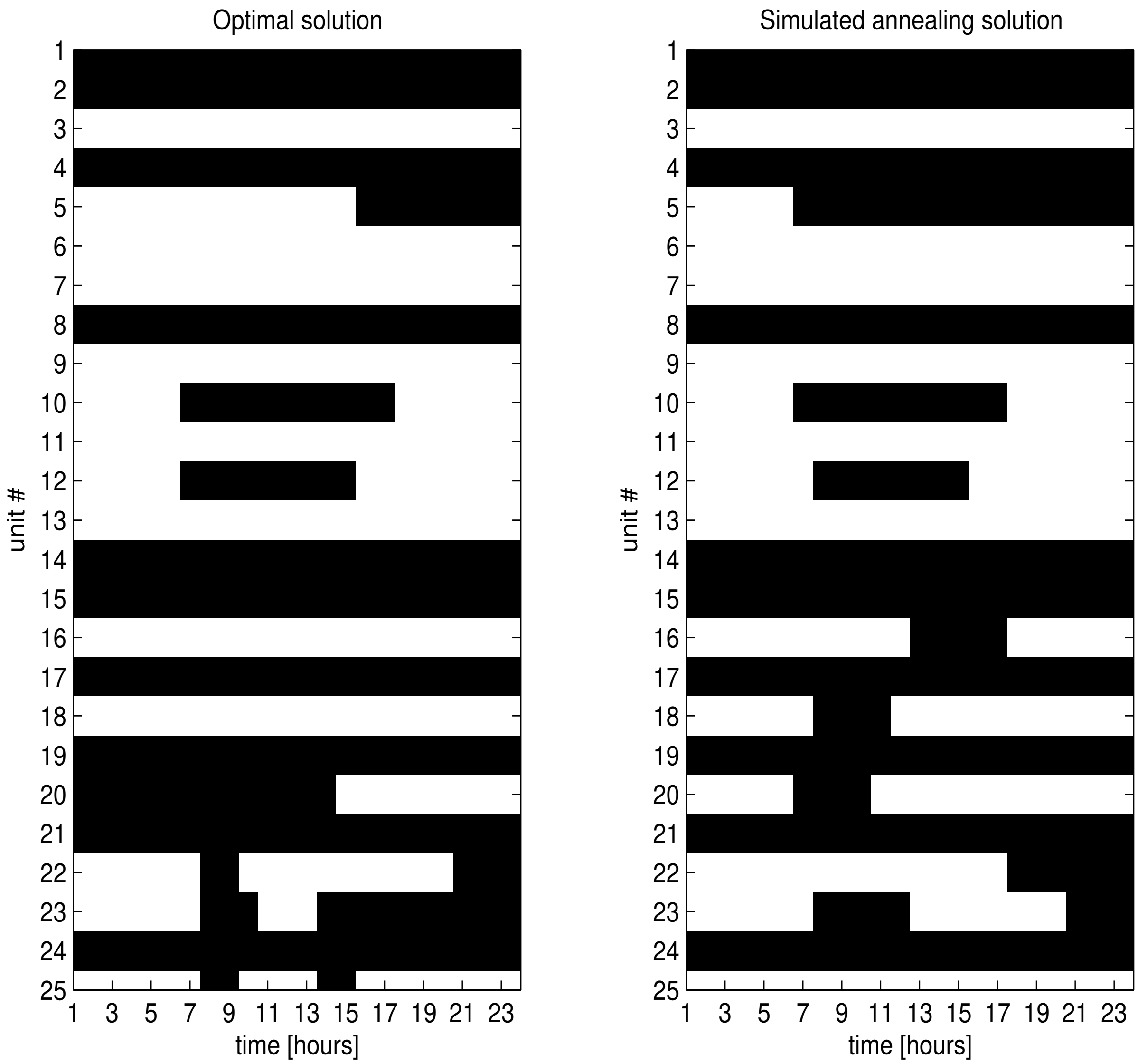

Figure 6.8: Unit activity plot of optimal and corresponding SA solution. The start-up cost is unchanged is this case. The black color means that unit is active and white means it is not active

In connection to figure 6.8, it is worth pointing out that units belonging to the rural area are number 8-11 and 15-25[4]. Units 1-7 and 12-14 belong to the urban area. It is interesting to see that both the optimal and SA solutions are more similar for the first 16 units than units 16-25. Units 16-25 are relatively small rural units and many of them are used as peak units. It seams that there are more possibilities of permuting

[4]units 8-11 are condensing units while units 15-25 are CHP units and heat only boilers

the production in these units compared to the larger units (1-16) were it is enough to activate one or two units to handle the peak demand. The 1500MW heat only boiler which was mentioned in connection with the capacity plot in the urban area has the number 12. Unit number 10 which is also opened during the peak hours is a 270MW condensing unit (for further studying of this figure please refer to appendix A).

Figures 6.7 and 6.8 show that many of the features in the optimal solution are captured by the SA solution. Units that are open or closed for the whole time horizon are almost the same in both solutions. The variation is manly in units that are open for short periods of time.

0.5x reduction in start-up cost

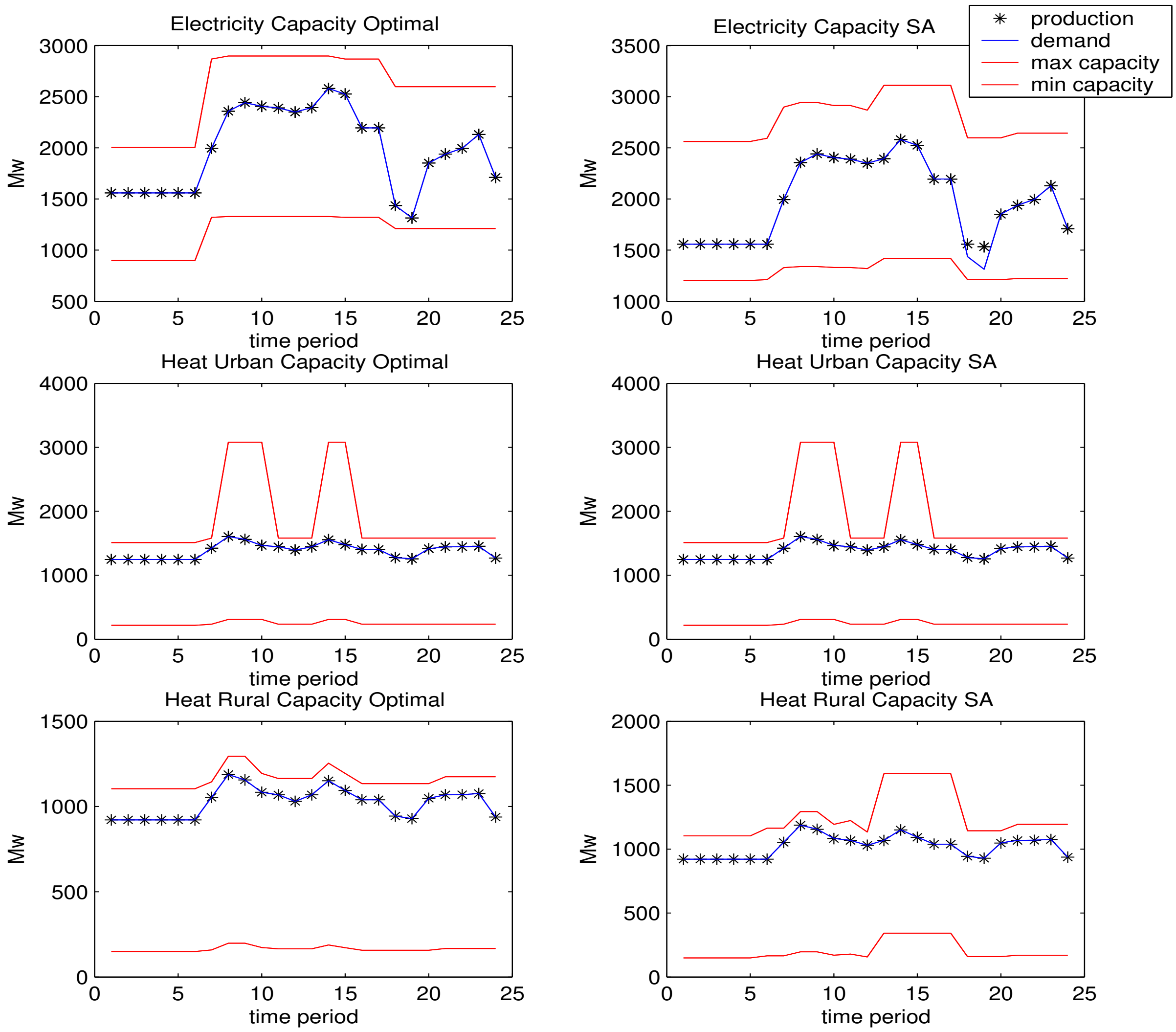

Figure 6.9: Capacity plot for a solution from test ♯6. The start-up cost is only half of its original value in this case.

One of the differences between the solution structure seen in figures 6.9-6.10 and the solution structure for the original start-up cost is that the 1500MW heat boiler in the urban area is now closed down for 3 hours between peaks in the heat demand. Since the start-up costs have been reduced, this possibility has become more economical.

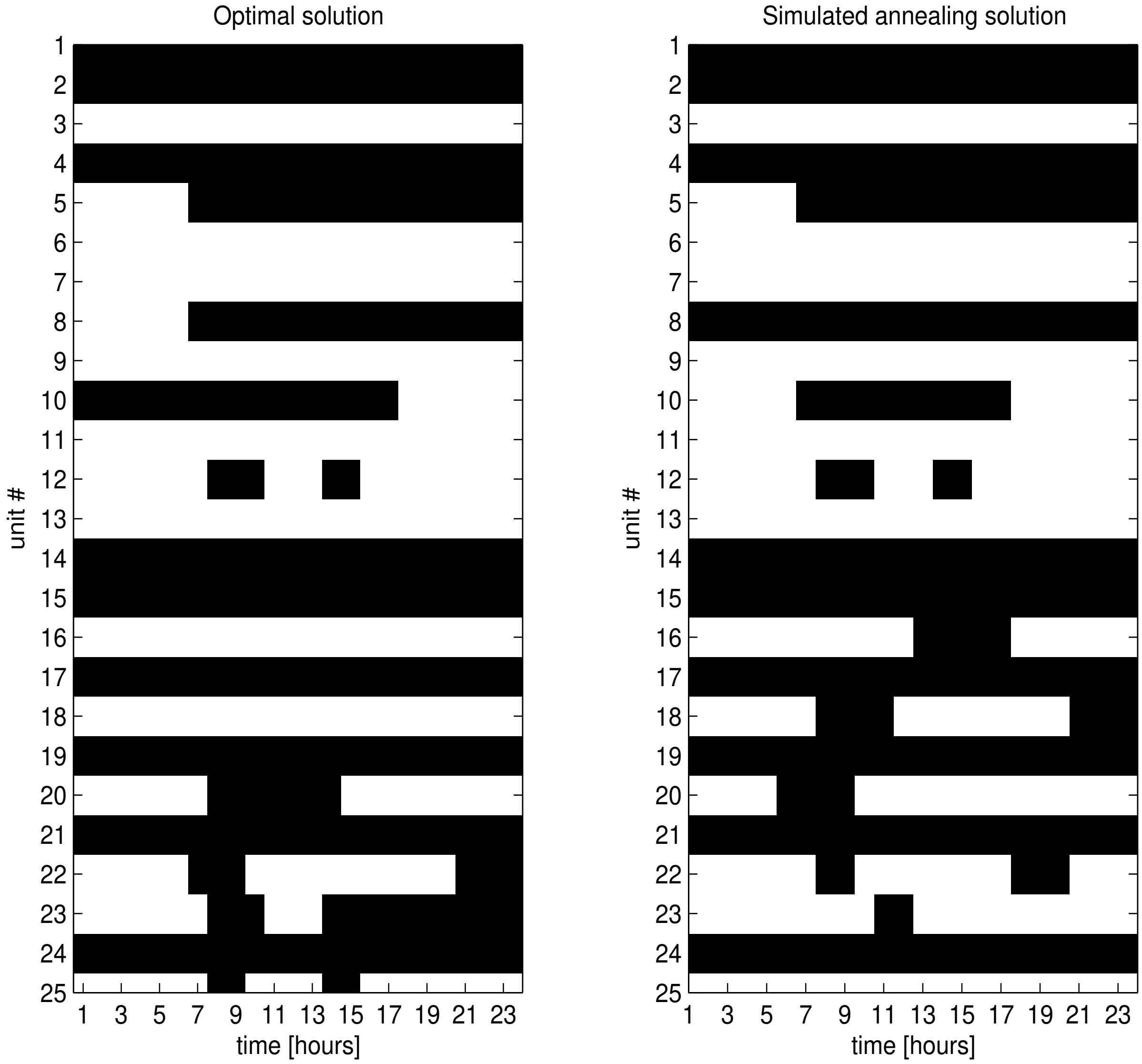

Figure 6.10: Unit activity plot of optimal and corresponding SA solution. The start-up cost is only the half of its original value in this case. The black color means that unit is active and white means it is not active.

As before, the SA solution in figure 6.10 gives to a large extent a similar solution structure as the optimal solution but there are some deviations. Comparing the SA solution for the original start-up cost to the SA solution here, it is interesting to see that the algorithm selects the same units as before but changes their production schedule slightly.

2x increase in start-up cost

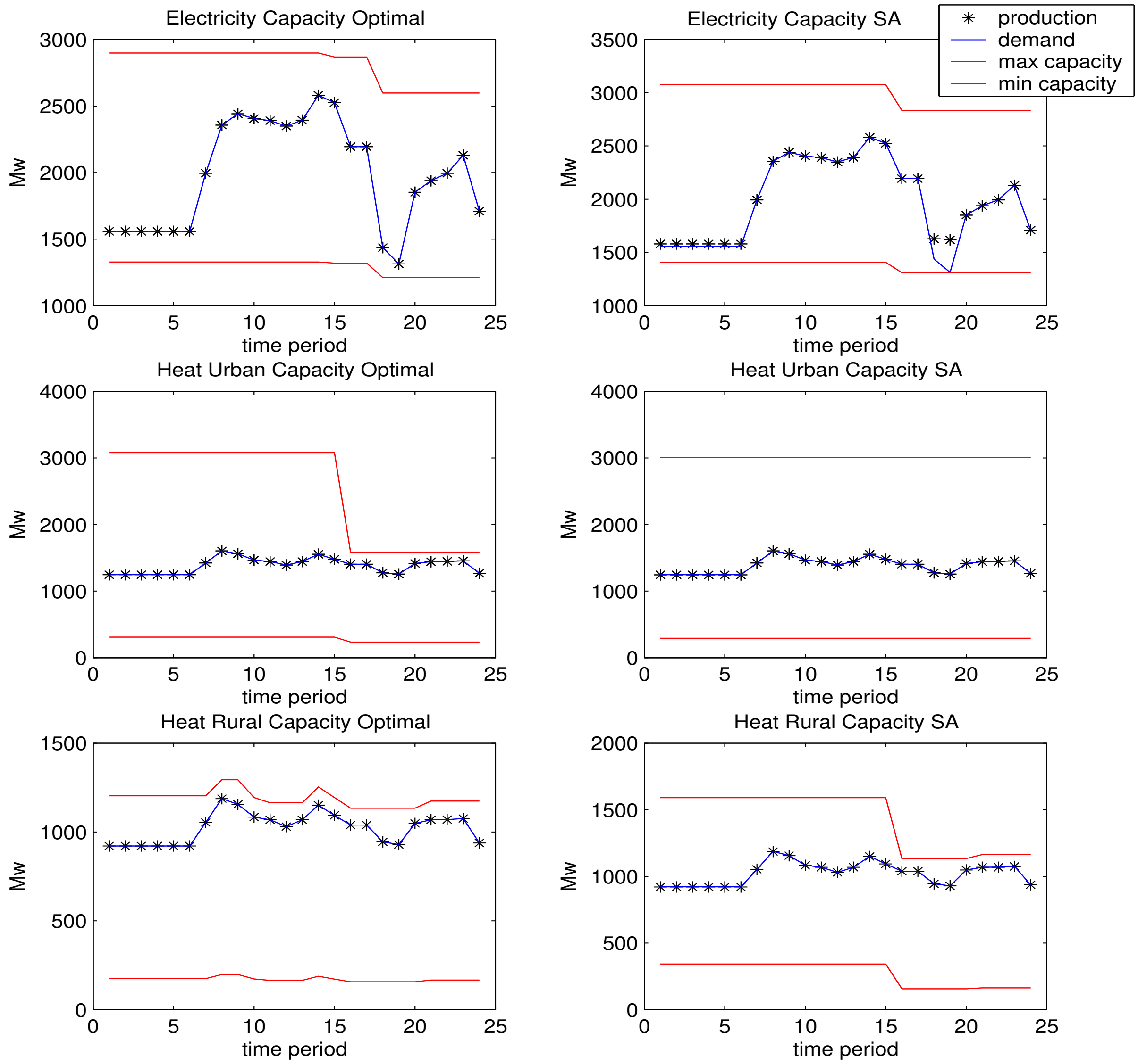

Figure 6.11: Capacity plot for a solution from test ♯1. The start-up cost has been doubled in is this case.

Looking at the optimal solution in figure 6.11, it can be seen that the doubling of the start-up cost has the effect that the 1500MW heat only boiler is now open from time period 1 and is only closed once the peak in demand is over in time period 17. The SA solution on the other hand keeps the boiler open for the whole time horizon.

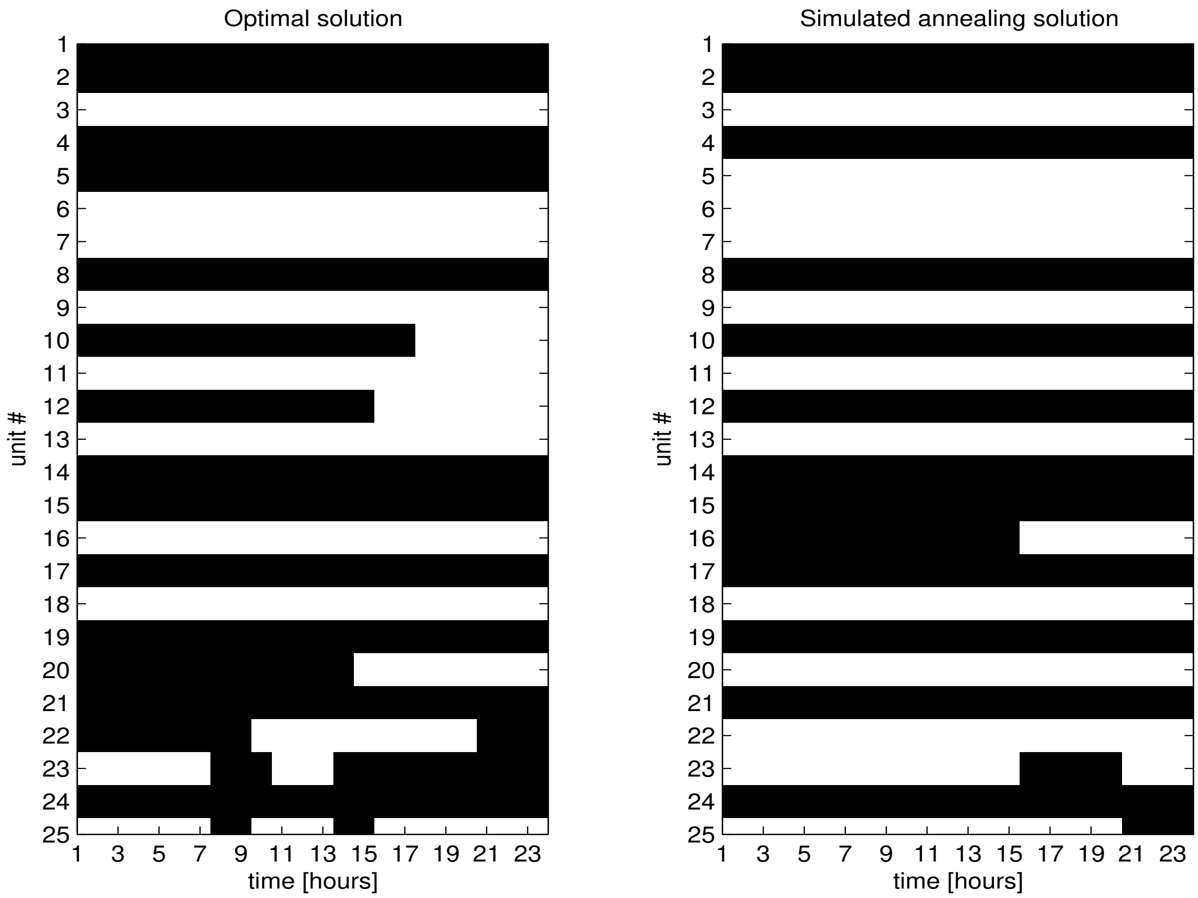

Figure 6.12: Unit activity plot of optimal and corresponding SA solution. The start-up cost has been doubled in is this case. The black color means that unit is active and white means it is not active.

The solution plot in figure 6.12 clearly shows that the increase in start-up cost is compensated for by limiting the number of start-ups throughout the production periods. The solution structure is now more rigid and units are either open or closed for many time periods in a row.

The solution plots for the three different start-up costs show that the SA algorithm responds to these changes in a similar way as the optimal solution does. The SA algorithm selects to a large extent the same units that are used in the optimal solution even though the schedule of these units can be different. The solution plots also show that even though the start-up cost is only a small fraction of the objective value it has significant effect on the solution structure.

The increase in variance and deviation from optimal value as the start-up cost increases, can be explained by figures 6.8 and 6.12. Both figures show that SA selects units that are not included in the optimal solution. This difference from the optimal solution naturally results in a higher objective value. Most of the units that are open for the whole time horizon are the same in both solutions and the units that differ are only open for a small number of time periods. When the start-up cost is increased, the number of time periods that each unit is open increases and the deviations between the solution structure obtained by SA and GAMS is therefore magnified. A unit which SA estimates feasible to produce in but is omitted in the optimal solution is kept open longer when the start-up cost is high. This is for example the case with unit 16 which is not selected in the optimal solution but its opening schedule increases from figure 6.8 to 6.12. This selection of "bad" units results in increase in both variance and deviation from the optimal solution when the start-up cost is increased.

6.3.2 The effect of minimum up and down times

The effect of the minimum up and down times was investigated in a similar manner as the start-up cost. The original minimum up and down times lie between 2 and 5 hours and all tests except one were made by increasing and decreasing these times by the same amount for each unit. The six different tests made are shown in table 6.7

min up and down	test 1	test 2	test 3	test 4	test 5	test 6
range	1	1-4	2-5	3-6	4-7	5-8

Table 6.7: Shows the order in which the minimum up and down times were tested.

In test ♯1 there are no minimum up and down constraints. In test ♯2 the original minimum up and down times have been decreased by one hour. This means that some units have no constriction on their opening and closing times. The original times are given in test ♯3. The time horizon used in the tests is 24 hours and each test is composed of 20 different runs of the SA algorithm. The optimal value for each test is calculated in GAMS for comparison. The results are shown in figure 6.13 where the results from each test are shown by using box plot and the optimal value is plotted below (marked with *).

Three conclusions can be drawn from figure 6.13. First looking at the objective values, it is evident that the minimum up and down constraints have very little effect on the optimal value (max difference is 0.1%). The second conclusion is that the minimum up and down times do not have effect on the performance of the SA algorithm as long as they are imposed on all units (tests 3-6). A t-test confirmed that tests 3-6 all have the same mean value. There is however a significant difference between tests 3-6 and test 2 (at 5% significant level). In test ♯2 some units have no minimum up and down times and that results in larger variance and deviation from the optimal value on average. This becomes worse once these constraints are no longer used at all in test ♯1.

There are two reasons for this larger deviation from the optimum. The first is that small minimum up and down times increase the size of the solution space which can increase the variance in the final solution. The main reason however, is probably due to the fact that the neighborhood algorithm used here is constructed specially to work with

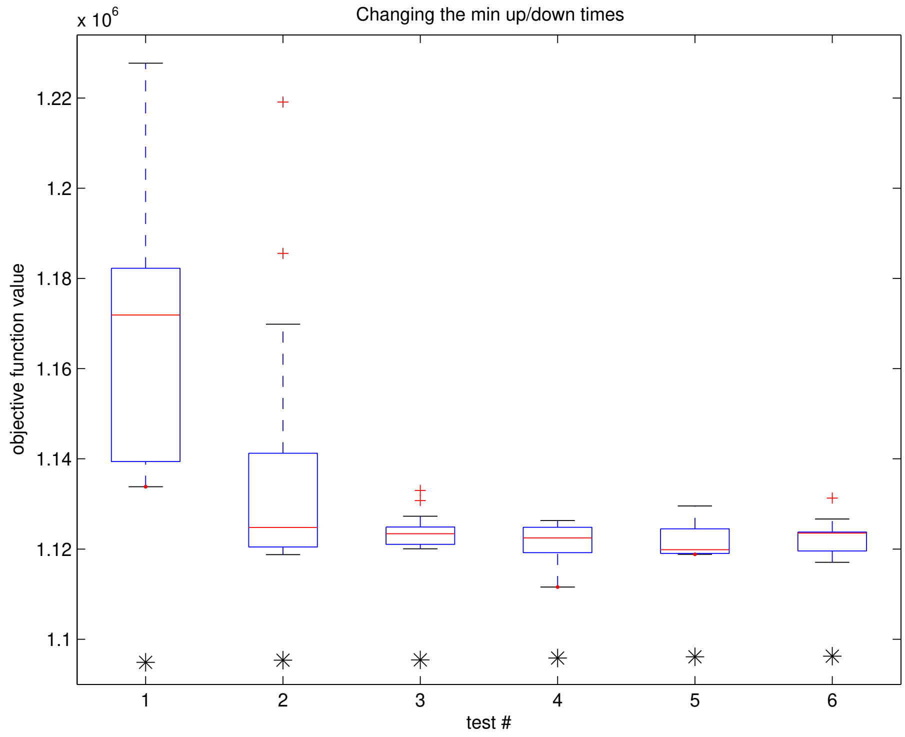

Figure 6.13: The figure shows the effect of increasing and decreasing the minimum up and down times from the original values (test 3). The only significant difference is for test 1 and 2. The optimal values are marked with an asterisk (*).

minimum up and down constraints. The case where these constraints are not used is simply not taken into account in the construction of the algorithm and another approach should be used in that case.

Finally it is worth mentioning that the effect of the minimum production capacity was also investigated. This was done by both increasing and decreasing the minimum capacity in all units by 10%, 30% and 50%. The results from these tests indicated that the minimum capacity did not have any effect on the performance of SA. Both the variance and the percent deviation from the optimal value was approximately the same as for the original value of the minimum capacity. No trend in the data was observed as found above for the start-up cost and minimum up and down times.

6.3.3 Increasing the time horizon

The effect of the time horizon on the solution quality is the subject of this section. The SA algorithm is tested on the problem for a time horizon of 24-96 hours (1-4 days). The SA algorithm is tuned separately for each time horizon and an optimal solution was calculated by GAMS. As for previous tests, the number of runs used in each test is 20. Table 6.8 shows the mean (μ), variance (σ) and the best solution (x_{best}) found for each test with the SA algorithm. The optimal solution from GAMS (x^*) is also given. The percent deviation from optimum for the mean, best and worst solution found is given in table 6.9.

hours	x^*	μ	σ	x_{best}
24	1.095×10^6	1.123×10^6	7×10^3	1.109×10^6
48	2.694×10^6	2.735×10^6	15×10^3	2.708×10^6
72	3.789×10^6	3.876×10^6	35×10^3	3.829×10^6
96	4.484×10^6	4.669×10^6	41×10^3	4.578×10^6

Table 6.8: Results for testing the problem for 1-4 days.

hours	%best error	% mean error	% worst error
24	1.3	2.6	4.0
48	0.5	1.5	2.4
72	1.1	2.3	4.7
96	2.1	4.1	6.1

Table 6.9: Percent deviation of μ and x_{best} from the optimal solution x^*.

It is a little bit surprising that the deviation of the mean value from the optimal solution is almost the same for a time horizon between 24-72 hours or (1.5-2.6%). When the time horizon is 96 hours the deviation on the other hand increases to (4.1%). An average error between 1.5-4.1% is quite acceptable for these tests if one takes into account that the worst case errors from the EDP heuristic can be as large as 4%. In these tests, the worst case errors are also relatively small and have the same order of magnitude as the mean error.

These results indicate that the EDP heuristic from chapter 3 is working quite effectively in estimating the quality of different unit combinations. It was expected that the use of the EDP heuristic might lead to larger errors in unit selection. It is however difficult to estimate the error contribution from the EDP heuristic and the SA algorithm separately

The main effect that can be seen from these tests is that the variance increases significantly as the time horizon is increased. If the SA algorithm had been tested for a longer time horizon one might perhaps get a better picture of when the solution quality starts to get worse. It was mentioned before that the unit commitment problem is usually not solved for a longer time horizon than 1-3 days. What happens for longer time horizon was therefore not considered to be of interest here and more effort was used in experimenting with other problem parameters.

Even though GAMS manages to calculate the optimal solution for the 4 tests above it is interesting to compare the computation times used by GAMS and SA. The computation times are shown in table 6.10

hours	SA	GAMS
24	15sec	20sec
48	28sec	420sec
72	31sec	720sec
96	35sec	960sec

Table 6.10: Computation time for GAMS and SA.

It is evident from these results that SA performs much better than GAMS with respect to computation time. Once the time horizon is larger than 24 hours the computation time explodes in GAMS. Even though the effect of the problem parameters were not tested in GAMS it was soon found out that GAMS was very sensitive to changes in demand and spinning reserve. Small variations in demand and spinning reserve had significant effect on the computation time. The computation time for SA increases only slightly as the time horizon increases. The periodicity of the solution could be the reason for this.

6.3.4 Increasing the number of units

In this section the number of units in the power production system is increased from 25 to 34 by dividing all units which have production capacity larger than 300MW into smaller units. Most of the units that are given in the *Balmorel* data with capacity larger than 300MW are in reality composed of two or more units. It is for example unlikely that the 1500MW heat boilers in the urban area are composed of one giant boiler and heat exchanger. The purpose of increasing the number of units in this way was to see how the SA algorithm would perform for a larger problem and if GAMS could still solve the problem to optimality. Based on the previous tests it was expected that the problem size was about to become too large for GAMS.

For each unit that was divided into smaller units, a random variation was added to the cost coefficients and the working area parameters to make the units distinguishable. The tests were made in the same way as for the time horizon test here above. The results are given in tables 6.11 -6.12.

hours	opt. sol/lower bound	μ	σ	x_{best}
24	1.031×10^6	1.057×10^6	9×10^3	1.047×10^6
48	2.562×10^6	2.588×10^6	5×10^3	2.579×10^6
72	$3.606 \times 10^6*$	3.672×10^6	22×10^3	3.635×10^6
96	$4.241 \times 10^6*$	4.414×10^6	21×10^3	4.386×10^6

Table 6.11: Results for testing the expanded problem for 1-4 days. Results marked with * in the first column are an estimated lower bound. The number of units is now 34.

GAMS was able to solve the problem to optimality for a time horizon up to 48 hours. The results for 72 and 96 hours are a lower estimate bound with a duality gap of 0.5% and

0.9% respectively[5]. Comparison of the results in tables 6.8-6.9 for the original problem and the results in tables 6.11-6.12, reveals that these results are very similar. The size of the % error and the variance is almost the same. The main difference is that the objective value is slightly lower now since certain units have been divided in to smaller parts.

hours	%best error	% mean error	% worst error
24	1.6	2.5	4.6
48	0.7	1.0	1.4
72	1.8	2.6	3.1
96	3.4	4.1	5.2

Table 6.12: Percent deviation of μ and x_{best} from the optimal solution x^*. The number of units is now 34.

For SA, the increase in the number of units does have a significant effect on the computation time which now increases from 17 to 73 seconds for a time horizon of 24 hours (see table 6.13). The SA algorithm has much shorter computation time than GAMS in all cases except for a time horizon of 24 hours. As before, the increase in time horizon has little effect on the computation time for the SA algorithm. In GAMS the computation time for 72 and 96 hour time horizon is measured until the GAMS calculation stops and a lower bound is returned.

hours	SA	GAMS
24	73sec	27sec
48	87sec	980sec
72	90sec	1380sec
96	94sec	1710sec

Table 6.13: Computation time for GAMS and SA. The number of units is now 34.

It is quite pleasing that the SA algorithm gives approximately the same results in this test using 34 units as it did with 25 units. The next logical step would have been to increase the unit number still further until some trend in the performance of SA would have been observed. Modifying the data in that way both with respect to GAMS and SA is however quite time consuming and there was unfortunately no more time available.

[5]the optimal solution is at best the value reported in table 6.11 and at worst 0.5% or 0.9% higher respectively

6.4 Comparison of SA and TS

This chapter is finally concluded by comparing the results obtained in this report to the results obtained by a fellow student who implemented Tabu Search for the same problem.

These two algorithms were compared by repeating the tests from section 6.3.3 for 1, 2 and 4 days. Each test is an average of 20 runs and the *worst*, *mean* and *best* solution found by SA and TS is plotted along with the optimal value. The objective value found in each test is divided by the number of days to allow better visual comparison. The results are shown in figure 6.14.

The figure shows that the results for SA have lower mean value in all three cases. The smallest difference is in the test for 2 days. In the tests for 1 and 4 days, the best solution obtained by TS is almost the same as the mean value obtained with SA. In general it was observed that both SA and TS obtained equally good solutions but TS had larger variance. The best solutions obtained by SA were also found by TS but the frequency was much lower.

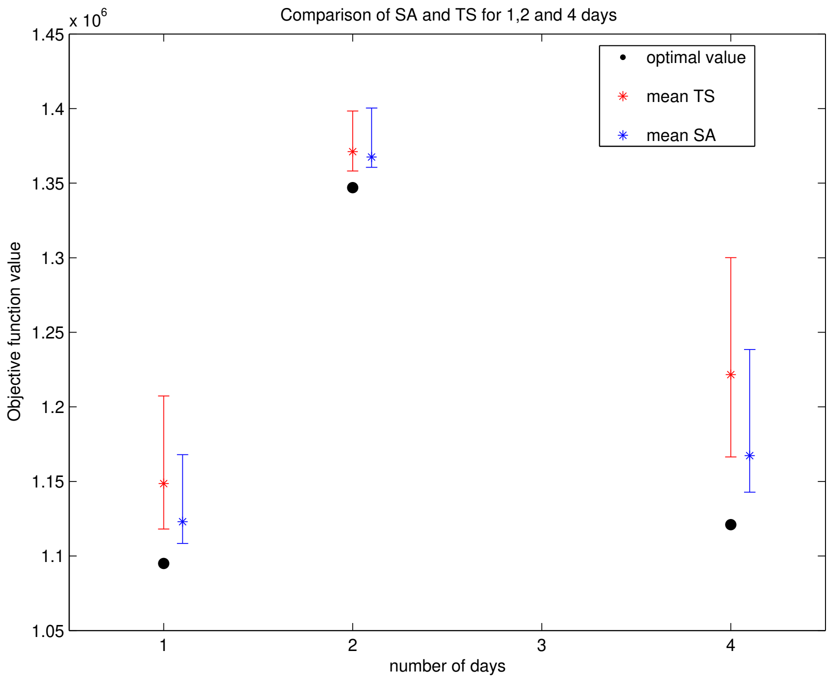

Figure 6.14: The figure shows the results for SA and TS for 1,2 and 4 days for the original system of 25 units. The *worst*, *mean* and *best* solution is plotted along with the objective value. Each solution has been divided by the number of days to allow better visual comparison.

A comparison of the iteration trajectory for SA and TS is shown in figure 6.15. The figure shows that TS finds good solutions in much fewer iterations than SA. It is only after about 50×10^3 iterations that SA starts to find equally good solutions as TS. It

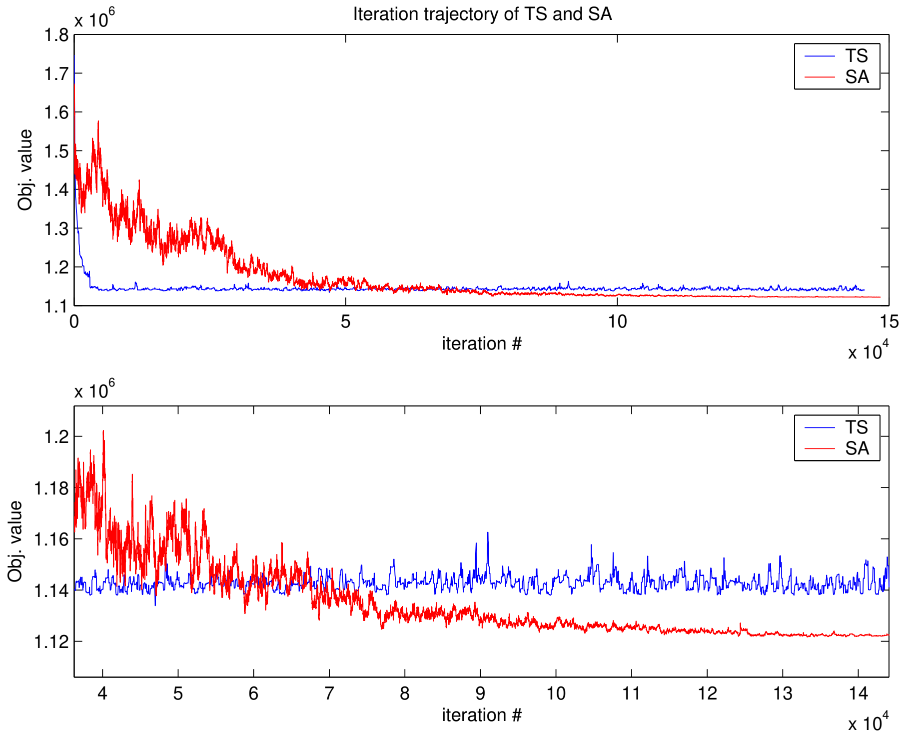

Figure 6.15: The figure shows the iteration trajectory for SA and TS. The lower figure shows the trajectory from iteration 40×10^3.

was generally found that TS located its best solution within 10×10^3 neighborhood generations[6] while SA needed at least 10 times that number to converge. For a time horizon of 24 hours the computation time of the two algorithms is however similar. The reason for that is that TS needs to copy the solution matrix when the best solution is updated and compare solution attributes to the tabu list. These are computationally expensive operations. Since SA makes only one change at a time to the current solution, the solution matrix only needs to be updated. That can be done quite fast.

With respect to the tests presented in this chapter, the following similarities between SA and TS were observed:

- They respond to changes in minimum up and down times, start-up cost, and minimum capacity in a similar way.
- The best solutions found are approximately the same.

The following differences were on the other hand:

- The computation time for SA was generally shorter.
- The number of iterations needed to locate the best solution was much shorter for TS than SA.

[6]remember that TS generates n solutions in each iteration and chooses the best one

- The final solutions from SA were 2-3% lower than the TS solutions on average.
- The final solution of TS was dependent on the initial solution.

The reason why SA produces better solutions on average can lie in the fact that solutions obtained by SA were found to be relatively independent on the initial solution. That reduces the variance between different runs of the algorithm significantly.

6.5 Conclusion

In this chapter the unit commitment problem was solved by SA for a time horizon of 24-96 hours using 25 and 34 units and the results compared to optimal or lower bound solutions calculated in Gams. The results obtained by SA were quite satisfactory with an average performance around 1-4% from the optimum, depending on the time horizon. The spread of the solutions obtained by SA are also relatively small with the largest deviation of 6% and smallest 0.5%. from the optimal solution. For a given time horizon the spread is never larger than $\pm 2\%$.

The testing of the SA algorithm showed that good results can be obtained by using a simple cooling schedule such as the geometric cooling function. Given a sufficiently high initial temperature, tests indicated that the performance of the algorithm was not strongly dependent on the initial starting solution. The conclusions that can be drawn about the effect of different neighborhood algorithms are rather limited since only two such algorithms were implemented and both use similar approach. It is however clear that the $N2$ algorithm which allows modifications to be made more freely than $N1$ works significantly better. The difference between the two algorithms is however only 1-2%. Even though the $N1$ algorithm was not used here its simplicity in implementation and its effect on computation time (compared to $N2$) can in some cases be an advantage. The comparison of the solution structures of the optimal and SA solution in figures 6.7-6.12 indicates that the neighborhood algorithm is working properly. The units that are selected and their opening schedules are to a large extent the same.

The experimentation with the 3 problem parameters: *Start-up cost*, *minimum up* and *down* times and *minimum capacity*, showed that increase in start-up cost has the most effect on the performance of SA. The reason for this effect of increased start-up cost is that the variance in the performance is magnified because of changes in the solution structure. There is nothing that indicates that the implementation of the algorithm is connected to this increase in variance. The effect of the minimum up and down times on the performance of SA is very little. The main conclusion is that another neighborhood algorithm should be used when the minimum up and down constraints are omitted in the problem formulation. A change in the minimum capacity of the production units had no effect on the performance of SA.

Regarding the problem size, the experimentation with GAMS indicates that the problem size is on the verge of being too large for an optimal solver to be used. One advantage of the heuristic approach, in this case, is that the computation time of SA is much shorter compared to GAMS. This is a valuable property when many experiments and testing need to be made with problem parameters in the underlying data. It is in many cases more desirable to get good approximate solutions fast rather than wait for long periods of time for an optimal solution.

The conclusions that can be drawn from the comparison of SA and TS is that the performance of both algorithms is quite satisfactory. There is nothing strange about the fact that one algorithm (SA in this case) performance slightly better. This is often the case with heuristic methods and it is not always easy to find an explanation. What is important is that the difference is small in this case.

Given that the heuristic results in section 6.3 are near optimal with respect to objective value and solution structure, it is concluded that the solution approach of using the EDP heuristic and the SA algorithm implemented, is a feasible option for solving the UCP for large CHP systems. There is nothing in the experimental results above that indicates that this solution approach will not work properly for larger systems.

Chapter 7

Conclusions

In this report the unit commitment problem has been defined and formulated for a CHP production system using data from the *Balmorel* project. Most of the data needed to formulate the UCP was available from the *Balmorel* project and only a few modifications had to be made. During the initial stages of this project it was concluded that the information available was sufficient to formulate the UCP in a realistic way. The model formulated in chapter 2 should represent a realistic power production system in the eastern part of Denmark. Some assumptions were made about the heat distribution system to increase the problem size.

The main purpose of this project was to implement a solution method to solve the UCP for a CHP production system. The method had to fulfill two requirements:

- The method should handle large production systems.
- The method should produce near optimal solutions in a reasonable computation time.

Because of the first requirement, a heuristic method had to be used. Simulated Annealing was chosen to solve the combinatorial part of the UCP because it is easy to implement and its solution process is transparent. Finding a method to solve the EDP was more problematic and much effort was spent on solving that problem.

To solve the EDP, linear programming was first considered. Several experiments revealed however that using this method was not feasible if a heuristic method was used to solve the combinatorial part of the UCP. The computation time of a linear solver was simply too long with respect to the number of iterations needed for a heuristic method. Instead, a fast heuristic method, based on a greedy approach was implemented to solve the EDP. The method had to be adjusted to each unit type in the production system. A ratio of production cost versus production output was used to compare production costs between CHP and non-CHP units. The method finds a solution by iterating in a greedy way from minimum capacity in a few iterations. The results obtained with this heuristic were surprisingly good. For a system of 25 units and 24 time periods the solutions from the EDP heuristic were never more than 4% from the optimal solution and in most cases much lower (less than 1.5% for the whole system). The computation time was only 1.2$msec$ for one time period which is two orders of magnitude faster than estimated computation for a linear solver such as Cplex.

In the implementation of SA, a neighborhood algorithm, that generated feasible solutions for the UCP, was implemented. Testing of SA showed that a simple cooling schedule could be used and that the final solutions of SA were not strongly dependent on the initial starting solution. It was also found that the basic version of SA worked properly and modifications such as using locally optimized SA or storing the best objective value did not improve the performance.

When SA was tested the results were compared to optimal or lower bound solutions calculated in GAMS. The results showed that the approach of using SA and the EDP heuristic to solve the UCP produces near optimal solutions in a short computation time. The problem size was 25-34 units and 24-96 time periods. The worst result obtained was 6% from the optimum and the mean error was 1-4% on average, depending on the size of the problem. The best results could be as close as 0.5% from the optimum. The computation time was 15-96 seconds depending on the problem size. This is considerably faster than the computation time in GAMS which could be as large as 960 seconds. For small problem sizes the computation times were similar. For larger problems the GAMS computation time exploded.

The effect of problem parameters was also tested and the main result was that the start-up cost has significant effect on the solution structure and the performance of SA. For large start-up cost the performance of SA is still satisfactory and the SA solutions respond in the same way to changes in start-up cost as the optimal solutions. In general the problem structure obtained with SA was to a large extent the same as the structure for the optimal solution.

Based on these results, it is concluded that

- The EDP heuristic is well suited when the quality of different unit combinations need to be estimated in a short computation time.
- The solution approach of using SA and the EDP heuristic is well suited to solve moderate or large UCP in a reasonable computation time.
- Experiments indicate that this solution approach is robust with respect to problem parameters.

There is nothing in the experimental results that indicates that this solution approach does not work for larger systems. The problem size is on the verge of being too large for GAMS and it is concluded that heuristic methods have to be used for larger problem instances. The solution approach presented here could be an alternative in those cases.

Bibliography

[1] C.-P Cheng, C.-W. Liu, C.-C. Liu, *Unit commitment by annealing-genetic algorithm*, Electrical Power and Energy Systems 24 (2002) 149-158.

[2] Chern-Lin Chen and Shung-Chung Wang, *Branch-and-Bound Scheduling for Thermal Generating Units*, IEEE Transactions on Energy Conversion, Vol 8, No. 2, June 1993.

[3] Chuan-Ping Cheng, Chih-Wen Liu and Chun-Chang Liu, *Unit Commitment by Lagrangian Relaxation and Genetic Algorithms* IEEE Transactions on Power Systems, Vol 15, No. 2, May 2000.

[4] Hanzheng Duo et. al., *A solution for unit commitment using Lagrangian relaxation combined with evolutionary programming*, Electric Power Systems Research 51 (1999) 71-77.

[5] Elin Hallgrimsdottir, *Optimizing Combined Heat and Power Plants in Copenhagen*, Institute for Mathematical Modeling DTU, exams project no. 19, 1998.

[6] Frederick S. Hillier and Gerald J. Lieberman, *Introduction to Operations Research*, sixth edition, McDraw-Hill Publishing Company 1995.

[7] Magnus Hindsberger and Rene Victor Valqui Vidal, *Tabu Search - a guided tour.* Control and Cybernetics, System research Institute, Polish Academy of Science. Vol 29, 3, 2000 pp 631-653.

[8] S.A. Kazarlis, A.G. Bakirtzis, V. Petridis, *A Genetic Algorithm Solutions to the Unit Commitment Problem*, IEEE Transactions on Power Systems, Vol 11, No 1, February 1996.

[9] Chun-Lung and Nanming Chen, *Direct Search Method for Solving Economic Dispatch Problem Considering Transmission Capacity Constraints*, IEEE Transactions on Power Systems, vol16. No 4, November 2001.

[10] A. H. Mantawy et. al., *Unit Commitment by Tabu Search.* IEE Proc. -Gener. Distrib.,Vol 145, No 1, January 1998.

[11] A. H. Mantawy, Youssef L. Abdel-Magid, Shokri Z. Selim, *Integrating Genetic Algorithms, Tabu Search, and Simulated Annealing for the Unit Commitment Problem*, IEEE Transactions on Power Systems, Vol 14, No 3, August 1999.

[12] A. H. Mantawy, Y. L. Abdel-Magid, S. Z. Selim, *Unit commitment by tabu search*, IEE Proc-Gener. Transm. Ditstrib. Vol. 145, No. 1, January 1998

[13] A. H. Mantawy, Youssef L. Abdel-Magid, Shokri Z. Selim, *A Simulated Annealing Algorithm for Unit Commitment* , IEEE Transactions on Power Systems, Vol. 13, No 1, February 1998.

[14] S.O. Oreao and M.R. Irving, *A combination of the genetic algorithm and Lagrangian relaxation decomposition techniques for the generation unit commitment problem*, Electric Power Systems Research 43 (1997), 149-156.

[15] Marc Pirlot and Rene Victor Valqui Vidal, *Simulated annealing: a tutorial.* Control and Cybernetics, System research Institute, Polish Academy of Science.vol 25, 1, 1996, pp 9-31.

[16] V. J. Rayward-Smith, I. H Osman, C. R. Reeves and G. D. Smith, *Modern Heuristic Search methods*, John Wiley & Sons Ltd, 1996.

[17] V. J. Rayward-Smith, I. H. Osman, C. R. Reeves and G. D. Smith, *Modern Heuristic Search Methods*, John Wiley and Sons, 1996.

[18] Colin Reeves (ed). *Modern heuristic techniques for combinatorial problems*, Oxford:Blackwell, 1993.

[19] Subir Sen and D P Kothari, *Optimal thermal generating unit commitment: a review*, Electrical Power & Energy Systems, Vol. 20, No 7, pp 443-451, 1998.

[20] Gersld B. Sheble and George N. Fahd, *Unit Commitment Literature Synopsis*, IEEE Transactions on Power Systems, Vol 9, No 1, February 1994.

[21] K. S. Swarup and S. Yamashiro, *Unit Commitment Solution Methodology Using Genetic Algorithm*, IEEE Transactions on Power Systems, Vol. 17, No. 1, February 2002.

[22] Tong, S. K. and Shahidehpour, *An innovative approach to generation scheduling in large-scale hydro-thermal power system with fuel constrained units* IEEE Transactions on Power Systems, 1990, Vol. PWRS-5(2), 665-673.

[23] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, *Introduction to Algorithms*, McGraw-Hill Book Company 2000.

[24] Jorge Valenzuela and Alice E. Smith, *A Seeded Memetic Algorithm for Large Unit Commitment Problems*, Journal of Heuristics, 8: 173-195, 2002.

[25] Laurence A. Wolsey, *Integer Programming*, John Wiley & Sons 1998.

[26] Suzannah Yin Wa Wong, *An enhanced simulated annealing approach to unit commitment*, Electrical Power & Energy Systems, Vol 20. No 5, pp 359-368. 1998.

[27] Stelios H. Zanakis and James R. Evans, *Heuristic "Optimization" why, when, and how to use it*, INTERFACES, vol. 11, No. 5, October 1981.

[28] F. Zhuang and F. D. Galiana, *Unit Commitment by Simulated Annealing*, IEEE Transactions on Power Systems, Vol. 5, No 1, February 1990.

Appendix A

Data from the problem formulation

Appendix B

C++ codes

B.1 The EDP algorithm

B.2 Simulated Annealing

Appendix C

GAMS related data

C.1 The GAMS $\rightarrow$ C++ connection

C.2 Solving the UCP with GAMS