Unit commitment for combined heat and power production

Using Tabu Search

Gauti Kjartan Gíslason

Spring 2003
IMM DTU

Preface

This M.Sc. thesis is the final requirement for obtaining the degree Master of Science in engineering. The work was carried out in the period from 1. February 2003 to 1. July 2003. The work was carried out at Operation Research section at institute for Informatics and mathematical modelling at the Technical University of Denmark. Supervisor in the project was Associate professor René Victor Valqui Vidal. Appendix for this report can be reached by contacting the supervisor or the author.

Special thanks goes to my supervisor for his guidance. Hans Ravn for his help in data mining in the start of the project. Bergthor Jonsson for his computer related assistant. Finally would I like to thank Thorahallur I. Halldorsson for his cooperation during the work of this project.

Lyngby, July 1st, 2003

Gauti Kjartan Gislason
s011078

Abstract

In this project the unit commitment problem for combined heat and power production is solved using heuristic methods. Unit commitment involves the scheduling of the on/off status of units in a power production system in order to meet load demand while satisfying operational constraints. The unit commitment problem is a combinatorial optimization problem. A sub problem of unit commitment is the economic dispatch problem , which determines the load dispatch for a given schedule. In this project the EDP is formulated as a linear programming problem.

A complicating factor in solving the EDP here 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 and variables is large. The solution approach used was to implement a fast heuristic method to solve the EDP. The heuristic was tested in comparison to an optimal solver and was found to produce near optimal solutions in all cases.

Tabu Search (TS) is used to solve unit commitment problem. A method that generates random feasible solutions for TS was implemented. TS was then tested on various sized problems and the results compared to solutions found using branch-and-bound. Finally, the performance of TS is compared to the performance of Simulated Annealing implemented by a fellow student working on the same project.

Keywords:

Unit commitment, economic dispatch, heuristic, combinatorial optimization, load dispatch, Tabu Search, combined heat and power.

List of abbreviations

- B&B: Branch and Bound
- CHP: Combined Heat and Power
- CO: Combinatorial optimization
- CPLEX: C and simplex
- DP: Dynamic Programming
- EDP: Economic Dispatch Problem
- GAMS: The General Algebraic Modelling System
- GA: Genetic Algorithms
- LR: Lagrangian Relaxation
- LP: Linear Programming
- MIP: Mixed Integer Programming
- $N(x_n)$: Neighbourhood of solution x_n
- $N'(x_n)$: A sub-neighbourhood of solution x_n
- $N(x_n, TL)$: A sub-neighbourhood of solution x_n
- $|N(x)|$: The number of solutions in N(x)
- OFF: Unit being inactive
- ON: Unit being active
- SA: Simulated Annealing
- TL: Tabu List
- TS: Tabu Search
- UC: Unit Commitment
- UCP: Unit Commitment Problem
- X: Set of all possible solutions

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 a 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
- Production 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 word combinatorial is reserved for problems in which the decision variables are discrete [28]. In this case 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 of 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 high as 80-90%. By contrast to conventional units, which 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 and the United States.

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). There has been quite a lot written about the UCP for conventional power production systems in the literature [5, 7, 9, 17, 18, 19, 21, 23, 24, 25, 35, 33, 29, 37]. Less has been written about the unit commitment problem were CHP units are included and the only one found was the masters thesis [13]. 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*[2] was started. The purpose of the *Balmorel* project is to support modelling 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]. The result of the project was 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

[2]for further information see http://www.balmorel.com

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

can range from days and up to several years. The model takes into account production output 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.

Because 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 with 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 originates from the *Balmorel* project and 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 computational time. The method of choice in this project is *Tabu Search*. To achieve reasonable computation time, a fast heuristic method that solves the EDP had to be constructed.

The assumption made in the model formulation are based on the quality of the *Balmorel* data that assumes linear production cost and unit constraints.

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

1.3 Outline

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

In *chapter 2*, the power production system analysed 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 then 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 in section 3.2. Section 3.3 gives more detailed technical description of the implementation. This section should be read by those who are interested in specific details of the implementation. The chapter is then concluded by presenting test results and performance measurements.

Chapter 4 gives an overview of some of the possible 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 is described and the advantages and disadvantages of each method are stated. The chapter ends by giving the argument for the selection of Tabu Search. In connection with the contents of other chapters in this report, the discussion of heuristic methods is most relevant (section 4.3).

The subject of *chapter 5* is the implementation of Tabu Search. Its main sections are about neighbourhood search strategy and tabu list implementation. First is a general idea of the neighbourhood search given then more detailed technical description. Other factors like initial solution creator, stopping criteria and calculation of the EDP are also mentioned. An argument is given for the selections made.

Experimental results are finally presented in *chapter 6*. The first part of the chapter focuses on the tuning and testing of Tabu Search. The performance of the method is tested on the problem formulated in chapter 2. Special emphasis is made on investigating the influence of parameters which 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.

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 Thorhallur Ingi Halldorsson (s010261). Chapters 2, 3, and parts of chapter 4 are for that reason to large extend 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. In the first section the properties of the power production system are analyzed. First, a short introduction on the general properties of combined heat and power (CHP) production is given. Then the power production systems are introduced. The assumptions made about the distribution network, demand and production cost are presented. Short descriptions on the different unit types are 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 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 the 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 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[1].

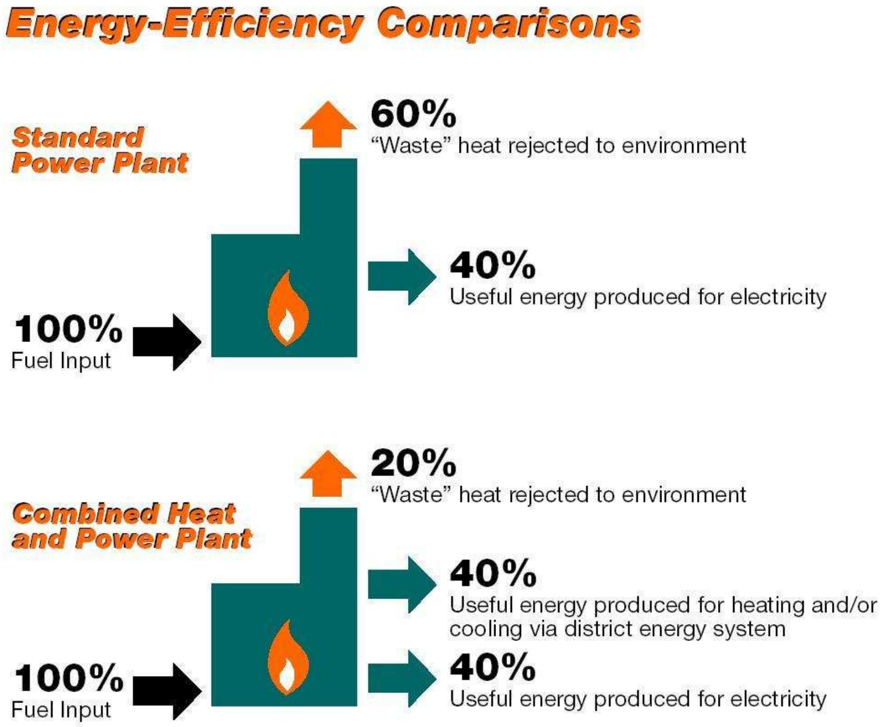

Figure 2.1: The figure shows the difference in efficiency between a conventional electricity producing unit and a CHP unit.

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. The distribution of heat is however 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.

[1]borrowed from the website of International District Energy Association: *www.districtenergy.org/chp.htm*

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 27 units[2]. 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 theses 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 the ON/OFF status of these units can not be controlled and 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 are 10 and 15 units are positioned in the rural area. The production capacity of the rural units is generally smaller than those 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 loss.

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 neighbourhood. Heat can therefore not be transported freely inside the whole area. 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 neighbourhood and since the heat demand is almost always non-zero these units are always active and therefore neither subjected to unit commitment, nor economic dispatch.

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 distributed between the rural 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

[2]Based on the data from the Balmorel project

area except heat only boilers[3] and only the condensing units in the rural area, or total of 14 units.

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

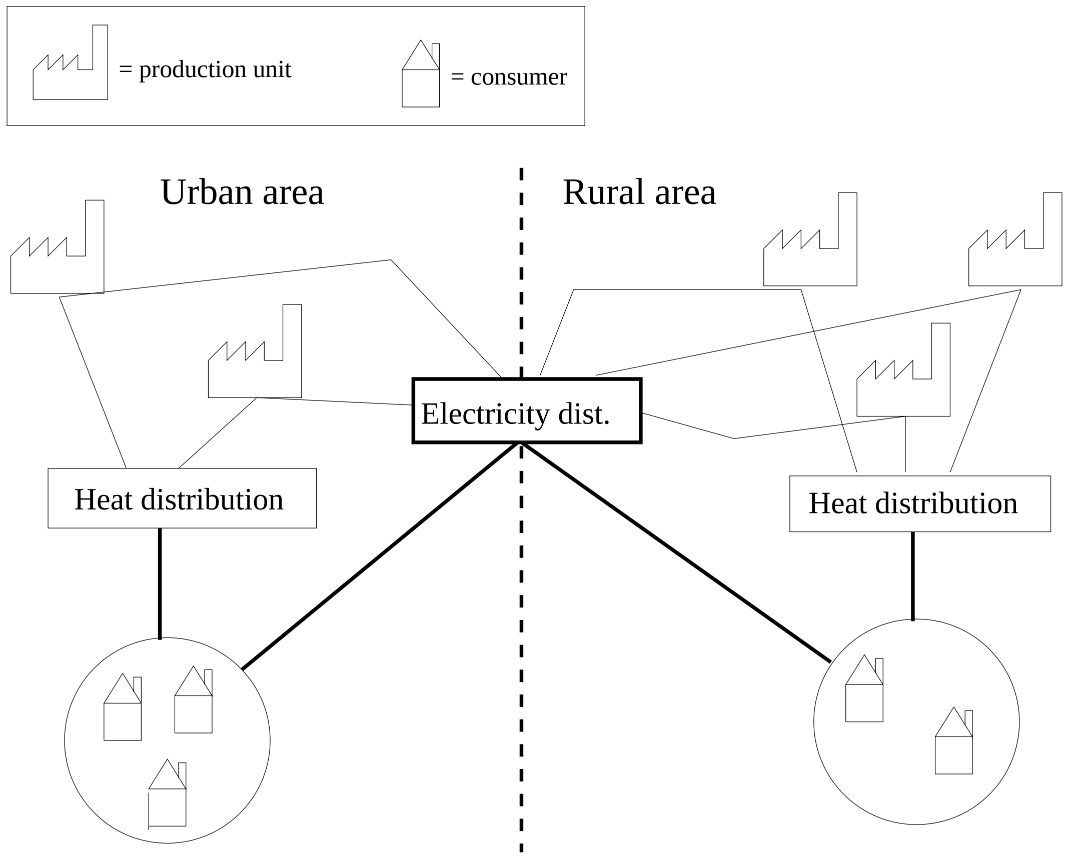

Figure 2.2: The figure shows the separate heat distribution network for the urban and rural areas. All units are connected to the same electricity distribution network

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

[3]see reason section 2.1.2.

both because units are subjected to malfunctions and the demand is difficult to predict accurately over long time periods.

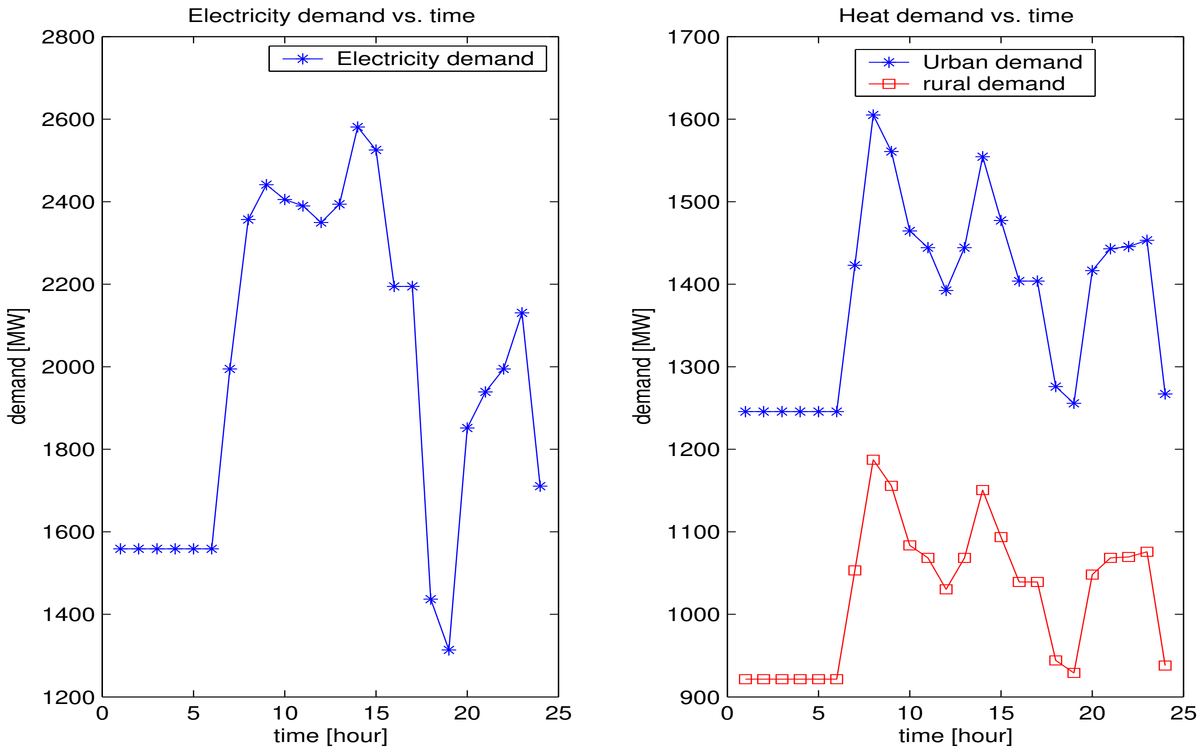

Figure 2.3: The figure shows the demand for electricity and heat (urban and rural area) for a typical winter day

In each time period the total production of heat and electricity must at least be equal to 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.

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 on 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 In contrast does 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 only one fuel type.
- *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 emission is highly dependent on the unit type and fuel 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 **??**.

2.1.2 Discussion of various unit types

In this section the properties of the five different unit types used in the formulation are discussed. The amount of electricity and heat feasible to produced in each unit is defined by its *working area*. The working area gives the physical boundaries for which the production in each unit must satisfy. The working area for each unit type is formulated and represented graphically. Every point (e, q), gives a combination of electricity and heat produced. If this point lies within the working area then it is a feasible working point. It should be noted that for all units the point $(0, 0)$ is also a feasible working point, which refers to unit being closed. One characteristic of the working areas is that it is not possible to produce heat or electricity continuously from $(0, 0)$ and there is an upper and a lower limit on production. The principles behind the production in each unit type are described shortly and their working area is both formulated and represented graphically. The minimum production capacity is not given in the *Balmorel* model and its value had to be estimated for all unit types. In this formulation are the working areas can be defined by a set of linear equations.

In the discussion that follows the following notation is used:

- t: index for time periods, $t = 1 \dots T$
- i: index for generation 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
- $F_c(i,t)$: fuel production cost in unit i at time period t, [EUR]
- $\gamma(i)$: fuel efficiency in unit i
- $f_c(i)$: fuel cost in unit i [EUR/MW]

The value of the variables e and q refers to one hour production. For example $e = 100$ means that the production in the relevant unit is kept at $100MW$ for one hour.

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 a schematic representation of a condensing unit.

The working area for a condensing unit 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).

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

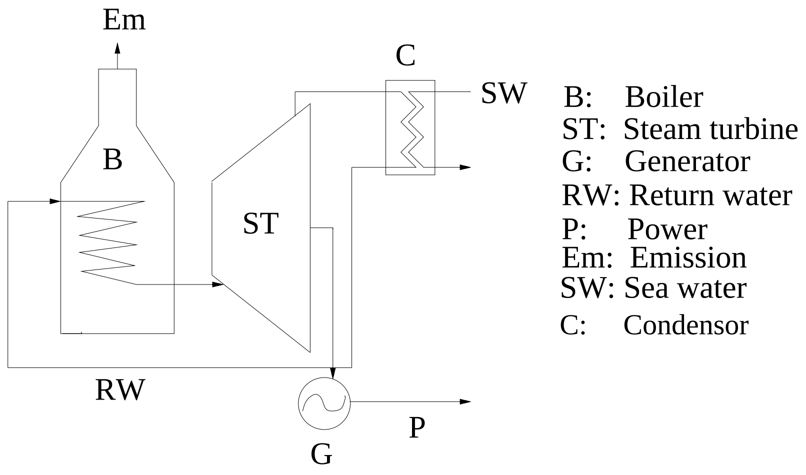

Figure 2.4: A schematic representation of a condensing unit

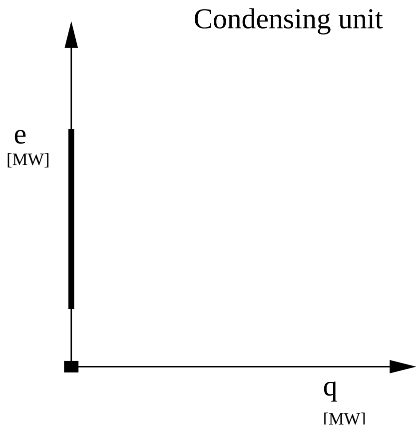

Figure 2.5: Working area for condensing units.

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. These units can produce heat almost continuously from zero. For that reason should they not be subjected to unit commitment. To make the problem more interesting (and larger) it was decided to have them subject to unit commitment but set their minimal capacity low.

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

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

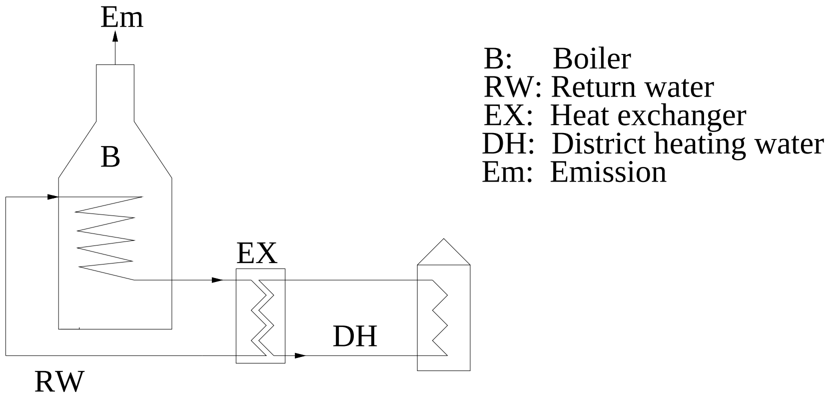

Figure 2.6: A schematic representation of a heat only boiler.

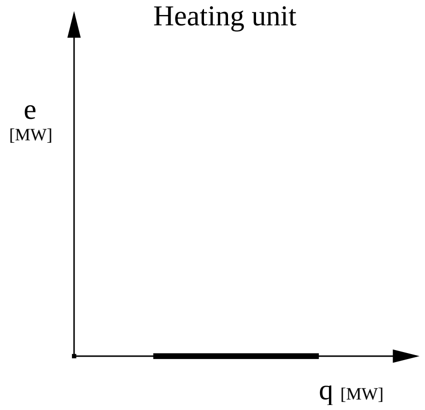

Figure 2.7: Working area for a heat only boiler.

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 unit work.

The working area for back pressure units is defined by equations 2.3, 2.4 and 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 $c_m(i)$.

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

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

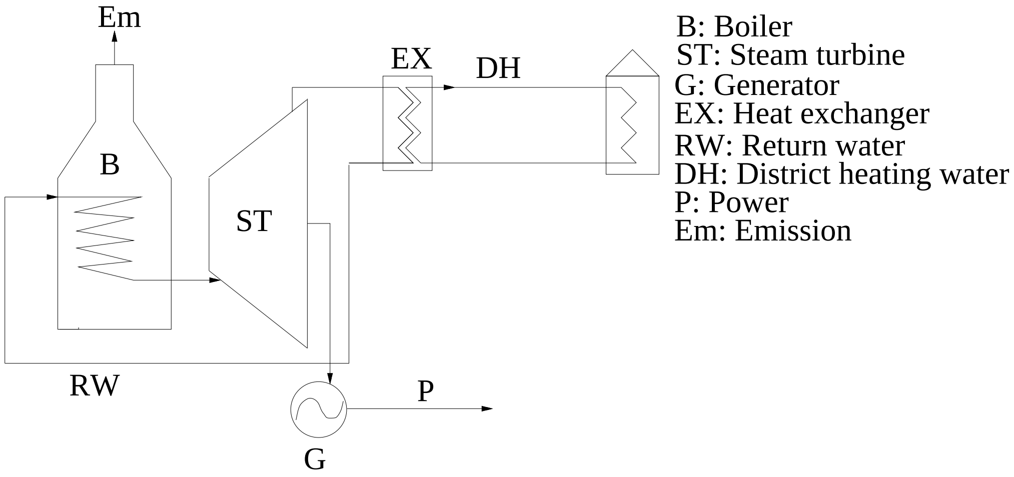

Figure 2.8: A schematic representation of a back pressure unit.

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

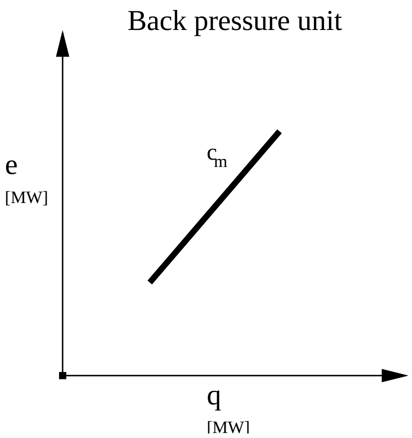

Figure 2.9: Working are for a back pressure unit

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.

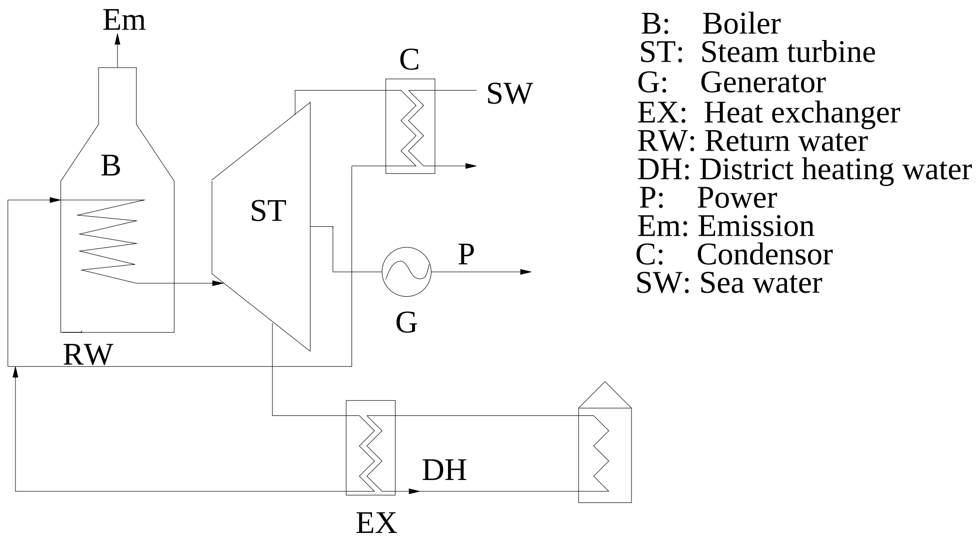

Figure 2.10: A schematic representation of an extraction unit

The working area for an extraction unit is defined by equations 2.6 to 2.10 and is shown graphically in figure 2.11.

$$q_{min}u(i,t) \leq q(i,t) \leq q_{max}u(i,t) \tag{2.6}$$

$$e_{min}u(i,t) \leq e(i,t) \leq e_{max}u(i,t) \tag{2.7}$$

$$e(i,t) \geq \alpha(i)u(i,t) - c_v(i)q(i,t) \qquad 0 < c_v(i) < 1 \tag{2.8}$$

$$e(i,t) \leq \beta(i) - c_v(i)q(i,t) \qquad 0 < c_v(i) < 1 \tag{2.9}$$

$$e(i,t) \geq c_m(i)q(i,t) \qquad q(i,t) \geq 0, \quad c_m(i) < 0 \tag{2.10}$$

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 2.10. 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, see fig 2.11.

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 2.9, points on or above the line given in equation 2.8 and all points on or left to the line given in equation 2.10

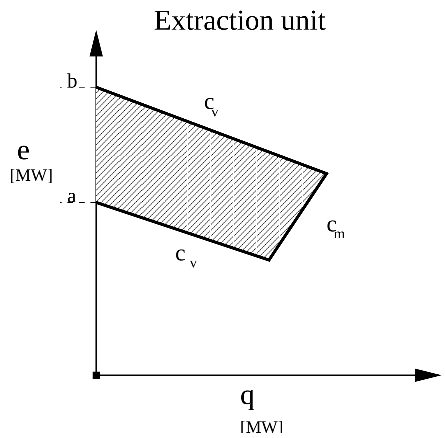

Figure 2.11: Working area for an extraction unit

Wind mills

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 [4]. 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 **??**.

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 time of 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 [33].

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

[4] given in the *Balmorel* data

- $T^{off}(i)$: 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 $\quad 1 < t < T$

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:

- Extraction units and heat only boilers are assigned 2 hours, because they can started up quite rapidly.
- 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.

The reason why the minimal up and down constraints are not used the first time period, is that it is assumed that the first time period belongs to a production pattern used previously. It is also assumed that the unit status of the last time period will remain the same in coming time periods, therefore are minimal up and down constraints not used for the last time period.

Peak units

Peak units are constructed in such a way that they have low start-up cost and can be started and closed rapidly but usually at the expense of relatively high production cost.

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 constraints). 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 differs between unit 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:

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

$f_c(i)$ 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:

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

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 in the amount of fuel consumed in the heat production:

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

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.
- $\alpha(i)$, $\beta(i)$: 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.

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

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

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

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

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 clearer.

The formulated model involves minimizing the objective function Z with respect to the three decision variables e, q and u, while satisfying all constraints. An 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 for a given combination of active units, with respect to these variables is called the economic dispatch problem. The EDP is a sub-problem of the unit commitment problem. The objective of the UCP, is therefore primarily to find best units to produce in at each time period.

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 [5] (equations 2.26-2.27)
- The minimum up and down constraints (equations 2.32-2.37)

2.3 Conclusion

This chapter has focussed on analysing the properties of the power production system 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 areas were introduced. All the data used for the model formulation originates from the *Balmorel* model and are 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.

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

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 algorithm [1] 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[2], 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[3] 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).

[1] An algorithm is any well-defined computational procedure that takes some value, or set of values, as a input and produces some value, or set of values, as a output [8].

[2] also defined in section 2.2.3

[3] exact methods: meaning methods that converge to an optimal solution

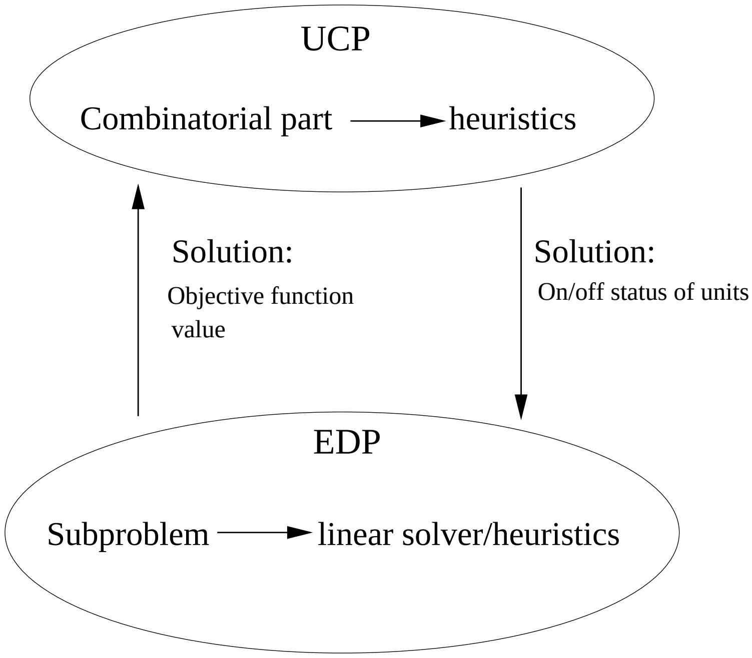

Figure 3.1: The figure shows the connection between the combinatorial part of the UCP and the EDP. When the combinatorial part is solved, solutions from the EDP are used to estimate the quality of different unit combinations.

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 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 that has hundreds of decision variables and constraints. Even though linear solvers work very fast, an execution time of 0.1 second, is not acceptable if the number of iterations needed is of magnitude 100.000. The effort used to solve 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?

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 when 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 algorithm 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 **??**

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 extendable for convex non-linear objective functions[4].
- The quality of the final solution must be sufficient to distinguish between the solution quality of different combinations of active units

[4] It is much harder to make a method that can handle non-convex objective functions

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* [7] 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 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 [18] and the results with respect to computation time has 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 considerably within a day and are difficult to estimate. This approach would therefore increase the complexity of the model. An easier way to compare the production costs between units is to use the following ratio:[5]

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

where

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

- $a(i)$ 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
- 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.

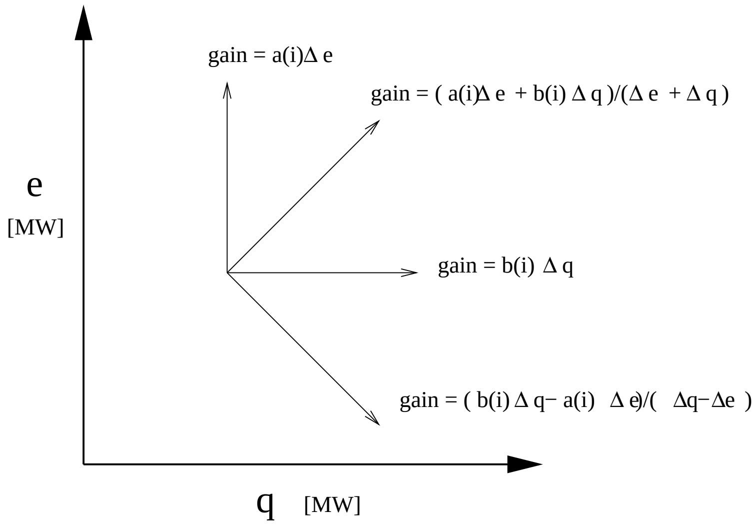

Figure 3.2: The figure shows the value of the *gain* for different search direction

In CHP units where e and q can be changed independently, the value of the *gain* is dependent on the search direction[6]. 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. 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)

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

- By limiting the search possibilities the method becomes easier to implement and computationally more simple (faster)

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. A decrease in production is in most cases forbidden. If a decrease in production would be allowed the whole process would be much more complicated. A 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 that holds the best possible *gain* in each unit. The initial production in each unit is its minimum capacity. In each iteration the algorithm chooses the unit with the lowest *gain*[7] 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 $e(i)$, $q(i)$ *in the search direction until demand is fulfilled or working area constraint is reached;*
        Update(L);
    Until all demand is fulfilled;

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

if excess production, then decrease that production if possible;
End;

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.

An important fact about the EDP is that time periods are independent. The start-up cost and the minimal up and down constraints binds time periods together in the UCP. But as these factors do not apply to the EDP and therefore are time periods independent. Advantage is taken on this and time periods are solved separately in the EDP heuristic.

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

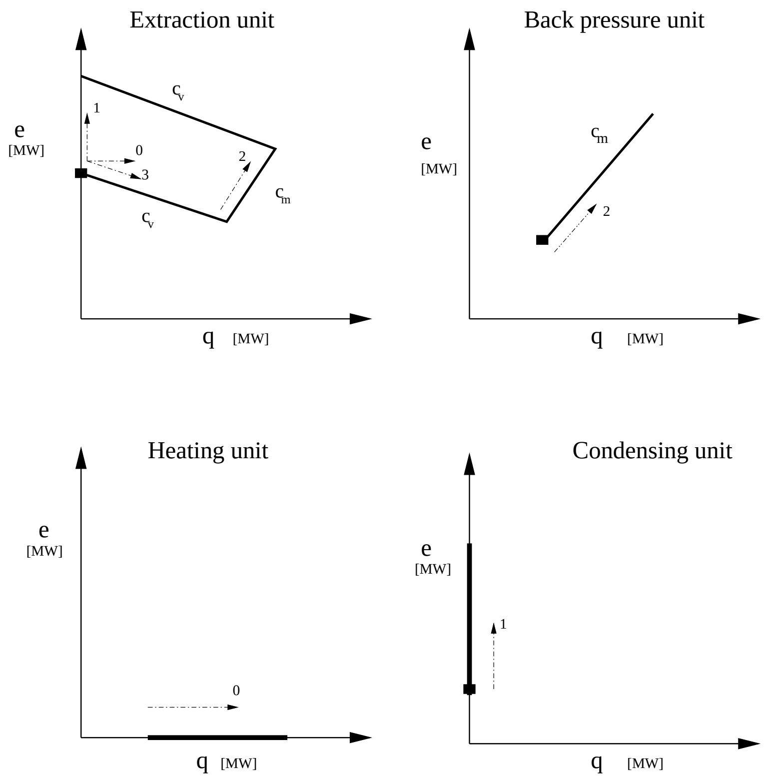

Figure 3.3: Allowed moves (search directions) for each unit type

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. In extraction units there is no point were electricity and heat are at its minimum capacity simultaneously. 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 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 up the electricity axis, 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 this method does not allow this *move*:

- It complicates the computations significantly
- It complicates the unit commitment problem with respect to estimating capacity

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).

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

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 reachable for each unit that is open for selection. If this condition is not fulfilled there 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 algorithm 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 ($move = 2$). This move is only selected when electricity demand has been met, then is decrease in electricity production not allowed along lower $c_v line$.

- 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 algorithm 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:

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 set is to 0, depending on which demand is met. This is done to eliminate the relevant 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 Heat Only()
9       if( unit = Back pressure )
10          do Back Pressure()
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 happen that production exceeds demand for either heat or electricity. This only happens 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 algorithm converges it checks if excess production can be decreased. This is what the repair function in line 17 of the algorithm 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 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 below:

pseudocode for the repair function

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. *remove out of* $\boldsymbol{U}$
5. *decrease production until boundary is reached or demand is met.*
6. *end while*

The construction and implementation of the algorithm has now been discussed in some details. The algorithm 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 to much weight in extraction units and the algorithm 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 to 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 to 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 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 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 the maximum demand is between 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 with 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 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 between 0-3%. The reason for 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 from 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 will have 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.

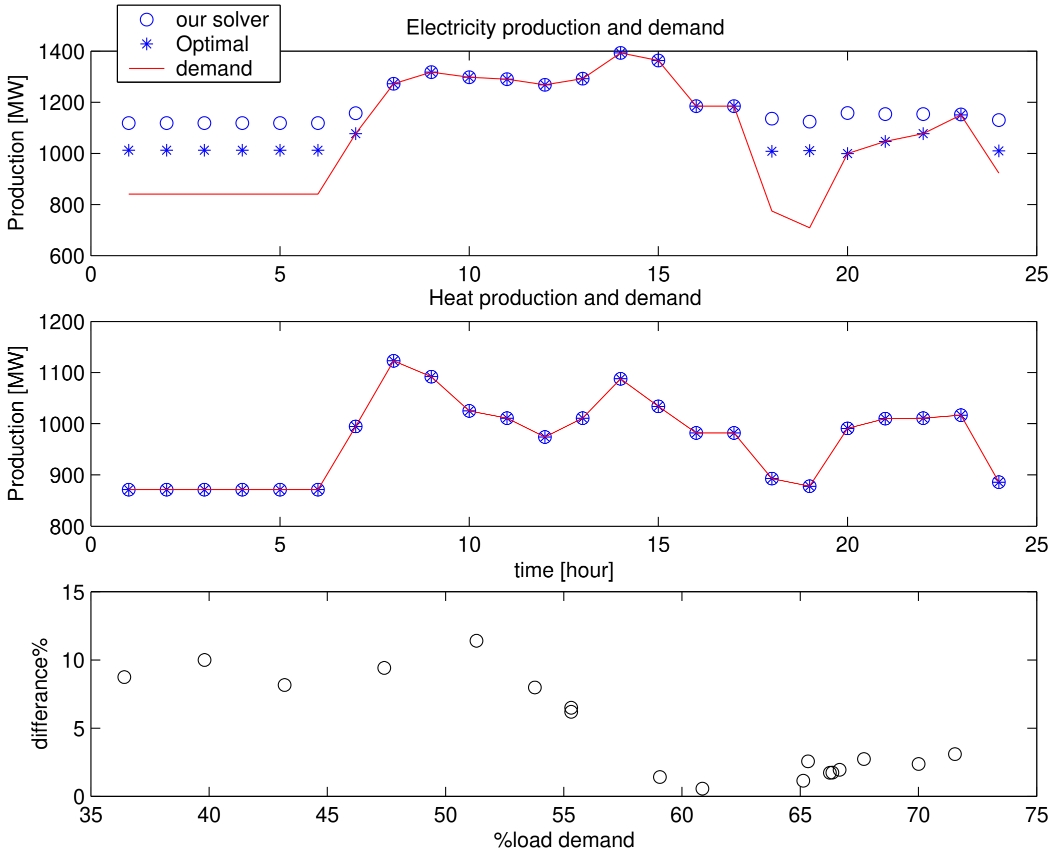

Figure 3.4: Comparison between the EDP-method and the optimal solution obtained with GAMS for extraction units. The top most figure shows the electricity production and the figure in the middle shows the heat production. The bottom figure shows the difference between the heuristic solver and GAMS with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

Back pressure units

Back pressure units have the characteristic 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, though usually electricity. In contrast to the test for extraction units, the largest errors are now obtained when the electricity load demand is high (then the error is $\approx 5\%$). A possible explanation for this can be 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. In this case, both solvers choose the same units for production when the demand is low. Excess electricity production can in this case not be

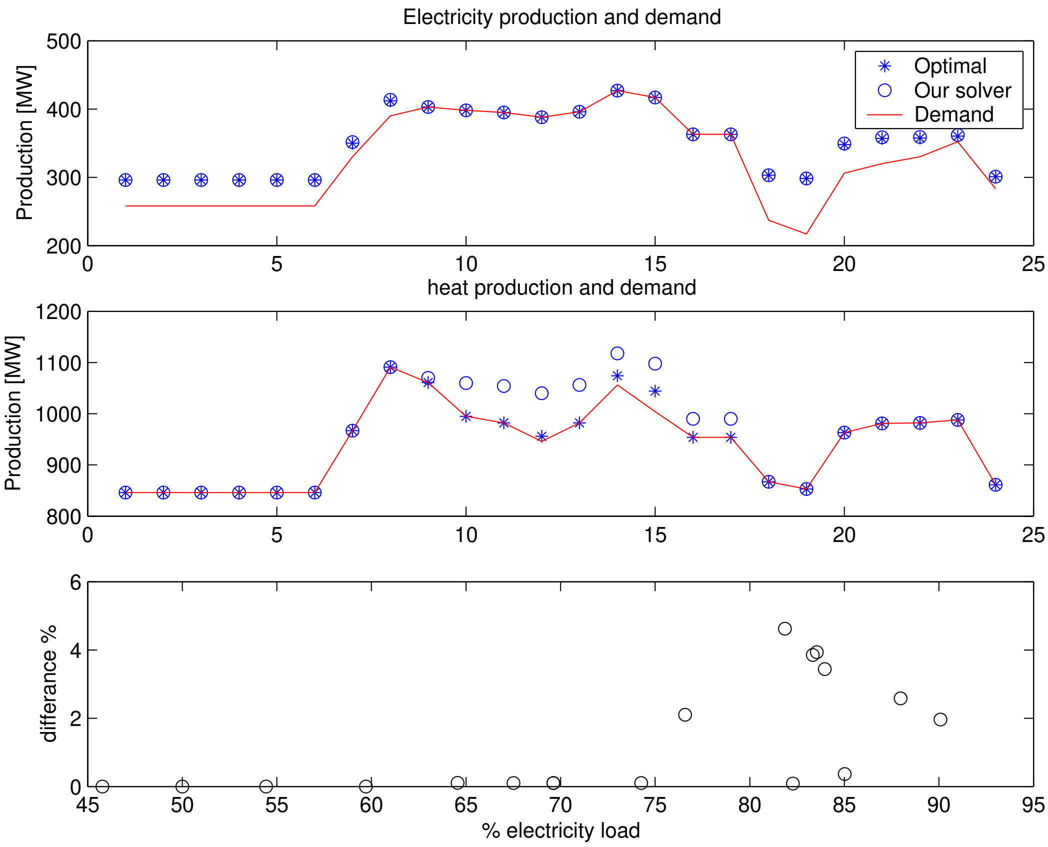

Figure 3.5: Comparison between the EDP solver and optimal solution obtained with GAMS for back pressure units. The top most figure shows the electricity production and the figure in the middle shows the heat production. The bottom figure shows the difference between the heuristic solver and GAMS with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

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.

Condensing units and heat only boilers

Both these units where 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 algorithm to iterate on the unit commitment problem repeatedly[8] (without start-up cost). A random unit combination is generated and objective value is calculated by both solvers. Then is a new unit combination generated (not so different from the current) if GAMS estimates it improving then is it approved and the EDP heuristic follows. If no better combination can be found, a new random combination is generated and the process is repeated. This was then repeated 4 times. The results are shown in figure 3.6 where the upper figure shows the greedy results for both the EDP heuristic and GAMS. The lower figure shows the percentage difference in the objective values found by the two solvers.

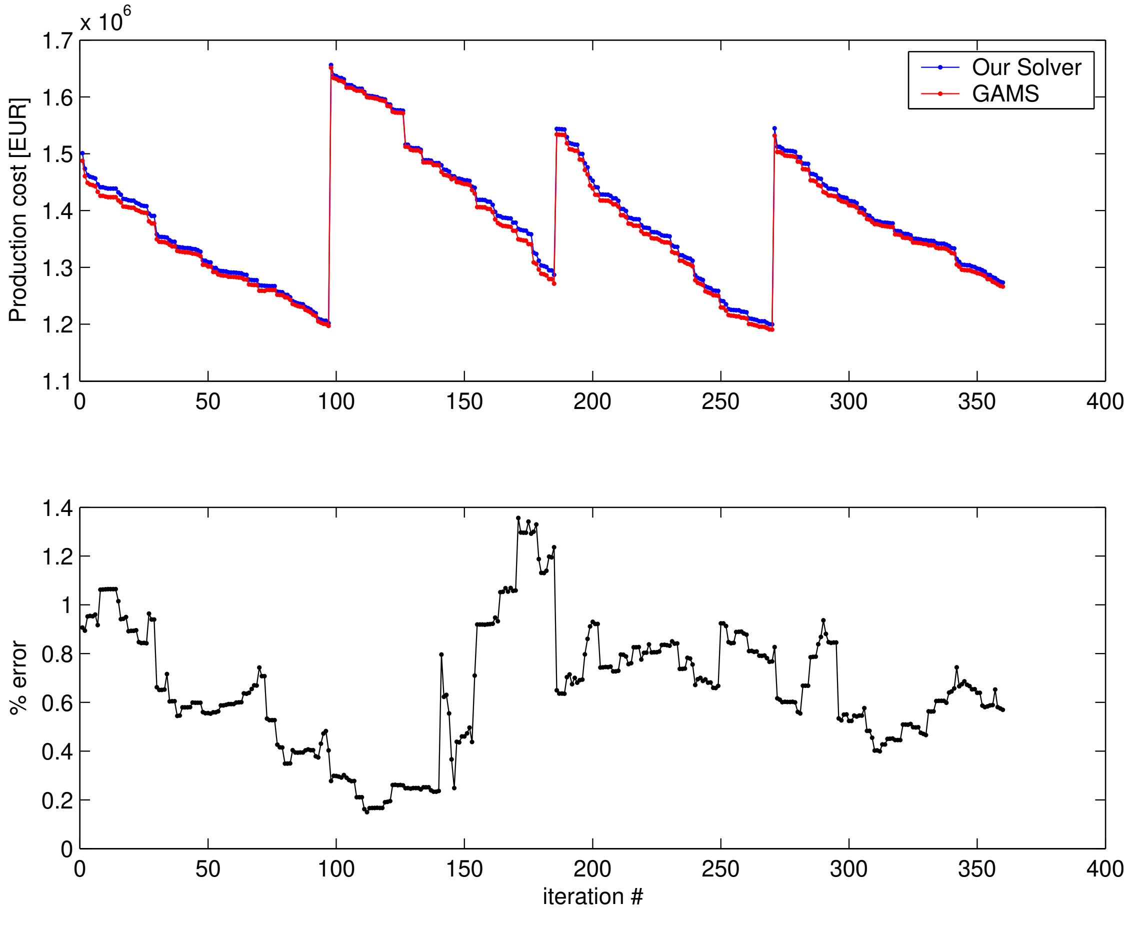

Figure 3.6: The upper figure shows the iteration trajectory from a descent algorithm used repeatedly (4 times). Each point corresponds to the objective value of a unit combination. The objective function value for both the EDP heuristic and GAMS is plotted. The lower figure shows the % error of the EDP heuristic, compared to the optimal GAMS solution.

[8]see section 4.3.3 for a description of a greedy algorithm

The maximum error for these measurements is 1.5%, which is a quite 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 with 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 form 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 the optimal solver. It however is difficult to prove if this holds for all possible cases.

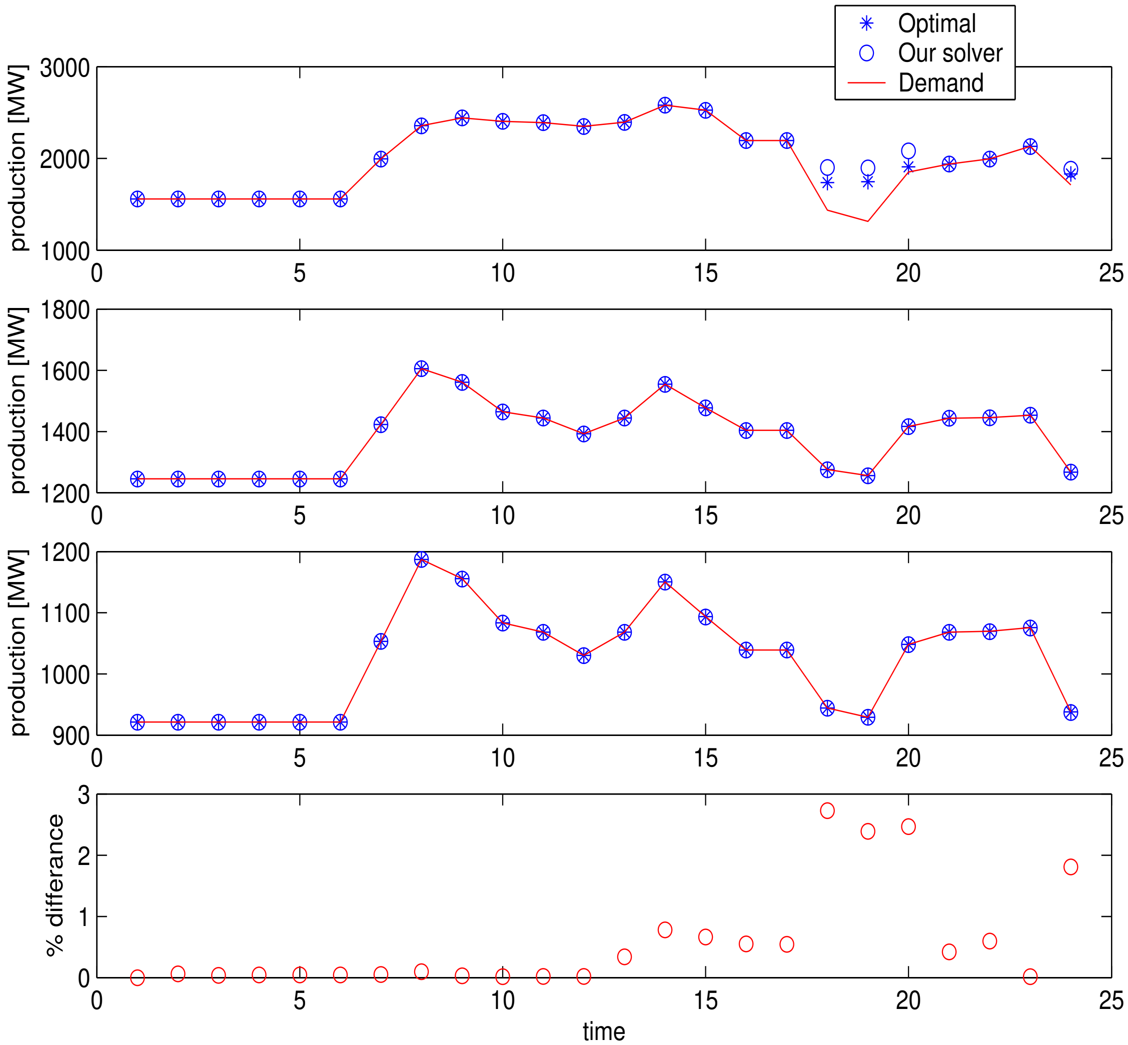

Figure 3.7: The figure shows an example of solution form the EDP heuristic for the whole system. Electricity, urban heat and rural heat production is plotted. The error with respect to the optimal production cost is given in the bottom figure.

Figure 3.7 shows the a solution obtained with the EDP heuristic for a final solution in the iteration trajectory in figure 3.6. This figure shows the same kind of plot as for back pressure and extraction units here above. 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 around 3% which is quite promising.

The test shown in figure 3.7 was made repeatedly when the EDP heuristic was tested after implementation, using thousands of different unit combinations. The maximum error observed for 1 time period was less than 4%. Even though a system of individual unit types can produce results with up to 10% error for 1 time period, 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 were 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 rather small as could be expected since the EDP is linear and advantages of that is taken in the calculation.

3.5 Conclusion

In this chapter the heuristic method implemented to solve the EDP problem has been described. Here the EDP problem is complicated by CHP plants which makes this problem much harder to solve. The method is based on greedy approach that tries to select the best units to produce in.

The EDP is a sub-problem of the UCP. For every solution to the UCP has the EDP to be solved. In that sense can the EDP be referred to as the objective function of the UCP. Without a good solver of the EDP problem would it never be possible to solve the UCP well. If there were considerable error in the solution of the EDP problem compared to optimal that error would increase when solving the UCP. Even though an extremely good method would be implemented to solve the UCP, quality of the final solution could never be ensured.

Due to this importance to find good solutions to the EDP it was originally the idea to solve the EDP with an optimal solver like GAMS. Unfortunately it was found out that this connection between the optimal solver and the programming language was too slow. Also it would be harder to extend the problem to non-linear if an optimal solver was used. It was therefore decided to implement a heuristic solver to solve the EDP.

The solver gave very promising results, and the objective value never was more than 3 % from optimal for the whole problem. This indicates that the solver could be used to evaluate the quality of a combination. An important result from the 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 positive property is that the solution time for the whole problem in the EDP heuristic was only around 1 ms. Which is 100 times faster than GAMS.

The idea with the solver was to make it extendable to solve problem with concave non-linear production cost function[9]. That is desirable as production cost is often dependent on the production status. The solver was originally implemented in Matlab[10]. The execution time in it was around 1 second for the whole problem. That was too slow so emphases was put on to implement the heuristic in another programming language (C++) where the execution time could be speeded up. That took some time, so there was no time to implement a solver, which was able to handle convex non-linear production cost function.

Not much change have to made to the solver so that is possible. Then instead of always increasing the production in unit until a boundary is reached then only a small step $(\Delta e, \Delta q)$ would be taken. In the pseudo code for the EDP solver in section 3.2 lines 6 and 7 would be replaced by:

6 $\qquad e(i) := e(i) + \Delta e_i;$

[9]it can not be proved that it would work non-convex cases.

[10]Matlab is a mathematical programming language

$$q(i) := q(i) + \Delta q_i;$$

Then after each increase the gains would be updated just like is done in the algorithm now. A large step $(\Delta e, \Delta q)$ would increase execution time but possibly increase error. The step length could also be changed dynamically like is done in [18] but that increases the complexity of the algorithm.

It can also be expected that the execution time of the EDP solver would increase drastically if the production cost was non-linear. The average production increase in the solver as it is now is around 280 MW and on average is there 11 iterations per time period. For example if the step size Δe and Δq was 5MW, sometimes both are increased but sometimes only one of them so at average the change would be around 7.5 MW. It could be expected that the execution time would increase by factor $280/7.5 \approx 40$. That indicates an increase in execution time, which is around 1.2 ms now for the whole system to 50 ms.

The results of using the heuristic method to solve the EDP and Tabu Search to solve the combinatorial part of the UCP are presented in chapter 6. Next chapter is devoted to discussion about possible solution methods for the UCP.

Chapter 4

Overview of solution methods for the UCP

Last chapter presented a heuristic approach to solve the EDP, which is a sub-problem of the UCP. The solutions obtained with that heuristic approach will be used to distinguish between the qualities of different unit combinations when the UCP is solved in chapter 6. The problem now is to find a solution method to solve the combinatorial part of the unit commitment problem. 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 [30] [31]. The solution methods discussed are divided into three groups:

- Exact methods
- Relaxation methods
- Heuristic methods

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

This chapter is divided into 3 sections where the methods belonging to each group are discussed. A short description of each method is given along with its advantages and disadvantages in connection with the UCP. At the end of this chapter an argument is given for selecting Tabu Search (TS). The implementation of TS is then discussed in chapter 5.

4.1 Exact methods

The obvious advantages of using exact methods 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, the problem 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. 3 different solution methods discussed here that have been used to solve the unit commitment problem to optimality. These are: *Extensive numeration*, *Dynamic Programming* (DP) and *Branch and Bound* (B&B). A short description of each methods 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, spinning reserve and minimal up and down times. For each feasible unit combination, the economic dispatch problem is solved and start-up cost 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 with 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 [30]. 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 [30].

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 then is 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, $1 \ldots i-1$. The formulation of the unit commitment problem must have this property (*Markovian property*) if for the method is to converge to optimality. Further reading about DP in general and use of it to solve UCP can be found in [14],[16] and [34].

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. The number of states 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 [30].

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 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 to 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 integer 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

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

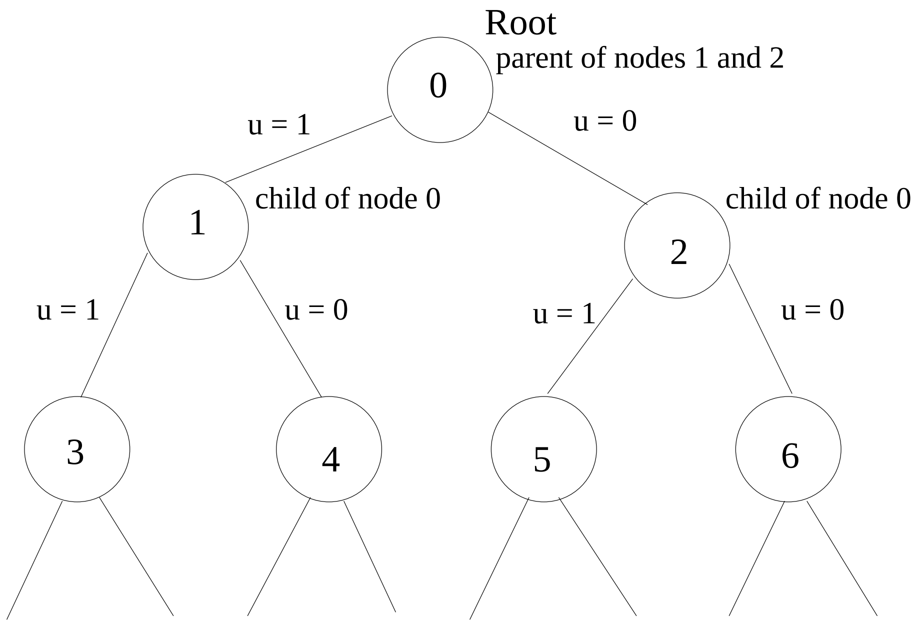

Figure 4.1: The figure shows the construction of the branch and bound search tree. In each node a constraint is imposed on a given binary variable ($u(i,t)$) that its value is either 0 or and 1. This results in two child nodes from the given parent node

solution of the original integer problem (for minimization problems). This is because the solution space for the linear problem includes all feasible solutions for the integer problem and all fractional solutions which can give better objective value than a pure integer solution. This bound then is 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. If not 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 [34] . 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 then 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

[2]if a lower bound for node i gives worse value then the incumbent integer 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

can be used to exclude all other nodes in the tree.

This is the principle that the branch and bound 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 integer values. Further discussion on the B&B method can for example be found in [34].

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 too many medium sized UCP see [6] and [13]. The method is quite extensively used in many commercially available solvers for mixed integer programming (MIP) problems and solvers are becoming more and more sophisticated. B&B is for example used in GAMS [3] when MIP problems are solved . 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. Another reason for failure can be that the size of the tree becomes too large for the computer memory.

4.2 Relaxation methods

Relaxation methods have that in common that they try to make the problem simpler by relaxing some of the constraints in the model. The optimal value found by solving the relaxed problem is always at least as good as optimal value of the original problem. Relaxing means that the solution space X is expanded to $relax(X)$. The original solution space X is a subspace of $relax(X)$. The optimal solution of the relaxed problem is also the optimal solution to the original problem if it is feasible to the original problem. Here, two different relaxations methods will be introduced, as possible solution methods to solve the UCP. They are Lagrangian relaxation and linear relaxation. These methods are actually very different but they have that in common that they both involve relaxation. Lagrangian relaxation relaxes some constraints and they are then penaltielized in the objective function. Linear relaxation alters the problem structure drastically as the integer part of the problem is relaxed. This means that the problem is actually solved as a large EDP were minimal production in all units is zero.

4.2.1 Lagrangian Relaxation

When solving the UCP the constraints that are summed over all units are very difficult. The reason why these constraints are so hard is because they do not only apply to one production unit but a set of production units. By relaxing these constraints and penaltielized them in the objective function, their difficulty can be reduced. Lagrangian

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

multipliers are therefore assigned to those constraints.

For the formulation in section 2.2 the Lagrangian function is given by equation 4.1. Here the constraints in equations 2.24, 2.25, 2.26 and 2.27 have been added to the Lagrangian function. These are the constraints for demand and spinning reserve. Solving this equation subject to the other constraints will be referred to as the master problem. This problem has a dual and its objective function is shown in equation 4.3. The difference between the optimal solution of the primal function and the dual function forms a duality gap, which can be used as a measure of the near-optimality of the solution. Equation 4.1 can be rewritten as equation 4.2. The term from equation 4.2 which is shown in equation 4.4 can be solved independently for every production unit and is called the sub-problem. The sub-problems are relatively easy to solve. The sum of all solutions for equation 4.4 for each generation unit subject to the relevant constraints gives a solution to the master problem. The reason why these sub-problems can be solved independently is because the connection between units no longer exists since those constraints have now been added as a penalty in the objective function. A more general description of Lagrangian relaxation can be found in [34].

The algorithm works as follows:
First starting values are selected for the Lagrangian multipliers which are used in solving the master problem. The master problem is solved by solving the individual sub-problems. The sub-problem is actually a DP problem. When a solution is found for the master problem (optimal e,u) it is fed to the dual problem. In the dual problem, new revised Lagrangian multipliers are found which and passed to the master problem again. This is repeated until the solution converges. The stopping criteria could be the size of the duality gap.

It can be difficult to find good values for the Lagrangian multipliers in the dual problem, since they tend to oscillate in last iterations, especially for larger problem instances. Solving the dual problem is therefore the most difficult step in the algorithm. When classical methods to update the multipliers fail (such as sub-gradient methods) other methods such as heuristics or variable matrix method can be used [30]. An example of connection of a heuristic method to find good multipliers can be found in [4],[7], [9], [23] and [33].

Master problem objective function

$$\begin{aligned}
L(e(i,t),q(i,t),u(i,t),\lambda,\mu) &= f(e(i,t),q(i,t),u(i,t)) \\
&+ \sum_{t=1}^{T}(D^e(t,a) - \sum_{i=1}^{N}(e(i,t))u(i,t)\lambda_t^e) + \sum_{t=1}^{T}(D^q(t,a) - \sum_{i=1}^{N}(q(i,t)u(i,t))\lambda(t)^q) \\
&+ (\sum_{t=1}^{T} D^e(t,a) + R^e(t,a) - \sum_{i=1}^{N}(e_{max}(i)u(i,t)))\mu(t)^e \\
&+ (\sum_{t=1}^{T} D^q(t,a) + R^q(t,a) - \sum_{i=1}^{N}(q_{max}(i)u(i,t)))\mu(t)^q
\end{aligned} \tag{4.1}$$

Where λ_t and μ_t are Lagrangian multipliers.

$$\begin{aligned}
&\sum_{i=1}^{N}\sum_{t=1}^{T}(f(e(i,t),q(i,t),u(i,t)) - \lambda^e(t)e(i,t) - \mu(t)e_{max}(i) - \lambda^q(t)q(i,t) - \mu(t)q_{max}(i))u(i,t) \\
&\quad + \sum_{t=1}^{T}(\lambda^e(t)D(t) + \mu^e(t)D(t) + \mu^e(t)R^e(t)) + \sum_{t=1}^{T}(\lambda^q(t)D(t) + \mu^q(t)D(t) + \mu^q(t)R^q(t))
\end{aligned} \tag{4.2}$$

Dual problem objective

$$\begin{aligned}
q(\lambda,\mu) =& min_{e(i,t),q(i.t),u(i,t)} L(e(i,t),q(i,t),u(i,t),\lambda(t),\mu(t)) \\
q(\lambda,\mu) =& \sum_{t=1}^{T}\sum_{i=1}^{N} f(e(i,t),q(i,t)u(i,t))u(i,t) + \sum_{t=1}^{T}\lambda^e(t)(D^e(t) + D^q(t) - \sum_{i=1}^{N} e(i,t))\sum_{i=1}^{N} q(i,t) \\
&+ \sum_{t=1}^{T}\mu(t)(D^e(t) + R^e(t) + D^q(t) + R^q(t) - (\sum_{i=1}^{N} e_{max}(i) + \sum_{i=1}^{N} q_{max}(i))u_{i,t})
\end{aligned} \tag{4.3}$$

Subproblem

$$\sum_{t=1}^{T}(f(e(i,t),q(i,t)u(i,t) - \lambda^e(t)e(i,t) - \mu(t)e_{max}(i)u(i,t) - \lambda^q(t)q(i,t) - \mu(t)q_{max}(i)u(i,t) \tag{4.4}$$

The advantages of using LR method are:

- Decomposition solution of the large-scale UCP due to the dual formulation.
- Easy extension to incorporate various constraints;
- Fast solution time
- It is more flexible than other approaches in handling different types of operating constraint [30].

The main disadvantage of the LR is associated with obtaining solution feasibility because of the dual nature of the algorithm. To try to avoid this problem with LR a revised method called augmented Lagrangian relaxation (ALR) was evolved.

Augmented Lagrangian Relaxation

As mentioned before the LR method can suffer from slow and oscillating convergence in the last iterations. In the Augmented Lagrangian relaxation (ALR) method an extra quadratic penalty term is added to the objective function using the same constraints as before. This will lead to faster convergence rate but does not affect the final optimal solution.

When the problem size gets larger ALR also has difficulty of finding good values for the multipliers in the dual problem. LR and ALR are therefore not practical for large problem instances. As mentioned above it is possible to use some other method (heuristics) with LR to make it handle larger problems. Examples of use of LR in general and use of it to solve the UCP can be found in [3],[4], [7],[9], [23], [29], [33] and [34].

4.2.2 Linear Relaxation

It is possible to solve the UCP entirely by linear programming if the integer constraints are relaxed. That means that the working area of different units is assumed to be continuous from zero, the resulting working areas are shown in figures 4.2 and 4.3. This implies that the problem is actually solved as a large EDP problem were all units are active at all times. Since the binary variable has now been relaxed the UCP is now a linear programming problem and can therefore be solved for much larger problem instances than the UCP. When the integer part has been relaxed, start up cost and minimal up or down times can not be used. Relaxing the integer constraint is a very crude approach as it is likely that the optimal solution to the relaxed problem is infeasible for the original problem. This can be a valid approach for larger problem instances. Then the problem becomes very sensitive to the quality of the data and even more sensitive to the data than the accuracy of the algorithm. An example of that is the *Balmorel* model.

Linear relaxation was used in master thesis [13] to simulate the production in the Copenhagen area. According to that thesis[4] is the difference in objective value found for relaxed problem and the original problem is always less than 1.5%. Infeasible solutions for the original problem where then repaired with some mechanism. On the other hand the optimal solution structure of the relaxed problem can be very different from the optimal solution of the original problem.

[4] page 60

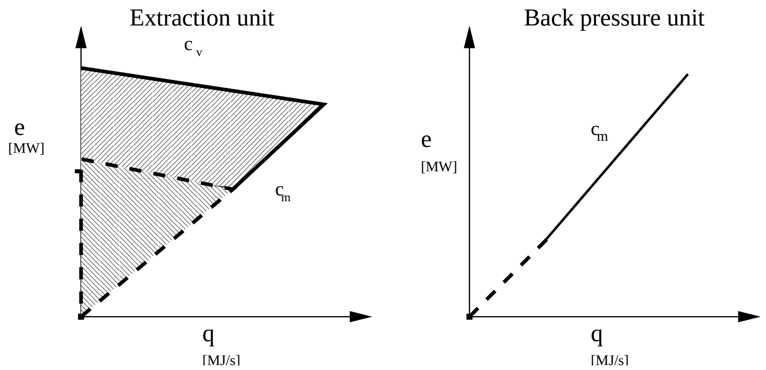

Figure 4.2: Working areas for Back pressure and Extraction units with linear relaxation. The linear relaxation is marked with a broken line.

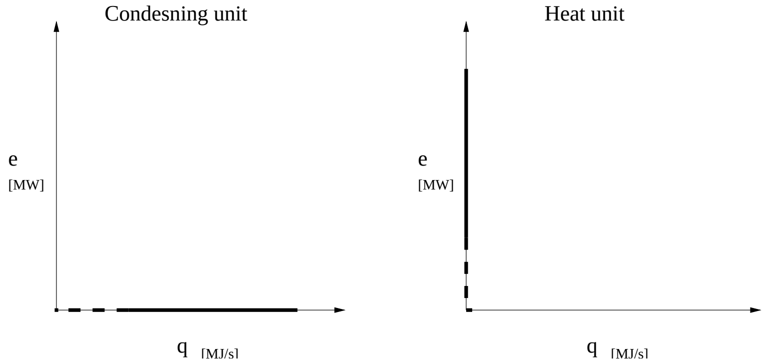

Figure 4.3: Working areas for Condensing and Heat only boilers with linear relaxation. The linear relaxation is marked with a broken line.

4.3 Heuristic methods

The word "heuristic" derives from the Greek word "heuriskein" and means "to discover" [36]. 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 some cases it is not possible to estimate how close to the optimum the final solution is. The exact methods discussed before guaranty optimal solutions but the price for that is often that long computation time goes into improving solutions that beforehand are quite close to the optimal solution. This is for example the case with branch and bound, which often finds a near optimal solutions after few iterations. It can then happen that the computer memory can not handle the size of the B&B tree and the algorithm brakes down before converging. In heuristic methods optimality is not given such a high priority but the emphasis is on finding good solutions in short time. This often results in the solution method being more simple and transparent than exact solution methods.

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

- They often give good solutions in rather fast 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 created specifically around a certain type of problem, for example like the EDP-heuristic in chapter 3, and some methods can be used to solve a family of different problems. The word metaheuristics is used for a set of heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be described as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adaptable to a specific problem. One characteristic of these methods is that they usually have a rather long execution time and due to the stochastic nature of most of the methods tends to give different results for two different runs. Examples of metaheuristic methods are priority list, *Genetic Algorithms* (GA) and the two local search methods *Simulated Annealing* (SA)and *Tabu Search* (TS). In addition will the heuristic methods priority list will be discussed. 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 details.

4.3.1 Priority list

The priority list method is a very simple heuristics and is often used in connection with other heuristic method, see for example [22]. In this method a priority list of units is created based on the value of a priority function which estimates the operation cost of each unit by some suitable method. The priority function could for example be the calculation of the *full load average production cost* [36]. Some other method or combination of other methods could also be used to estimate the operational cost. The list is then used to determine the on/off status of each unit in a greedy way such that demand and reserve constraints are satisfied. Since it is quite easy to estimate an operational cost of a unit, this method can be applied to problems where the number of units is relatively large. The disadvantage of this method however is the poor quality of solution which is quite often far from optimal. The main advantage of this method is that it is easy to implement and works fast. This method could be used to find a good starting solution for some other heuristic method.

4.3.2 Genetic algorithms

Genetic algorithms are search techniques based on the principles of natural selection. GA operates as an iterative search procedure on a set of candidate solutions of fixed size. Each candidate solution is usually encoded as a binary string and the encoded solution is called chromosome. Associated with each chromosome is a fitness value which is a measurement of the quality of the chromosome. An example of a fitness value for a chromosome might be the objective function value corresponding to the decoded chromosome. New solutions are generated from old solutions by using genetic operators. The two main operators that are used to generate a new solution are:

- *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. An example of crossover can be seen in figure 4.4. The procedure is as following: Every solution is given a value P (sum of all P is 1) value which is larger for solutions with good fitness value than those with worse. Then a uniformly random number between 0 and 1 is selected and chromosomes paired together depending on this random number.

- *Mutation*
  Randomly chosen bits of offsprings are changed from 0 to 1 and vice versa. Mutation injects new properties into the current population and prevents the algorithm to get stuck in a local minimum. An example of mutation can be seen in figure 4.4

A short description of the GA is as follows:
In the beginning of the algorithm, M solutions are randomly generated, their fitness value is then calculated and the solutions are encoded into chromosome. Individual solutions are next selected in pairs to form offsprings with a certain selection probability. The probability is higher for individuals who have high fitness values. After 1 to M new individuals have been generated they are put into the set of solutions and same number of worst solutions are deleted. This procedure is repeated until the population converges. Many generation are needed for convergence to a near optimal solution.

Figure 4.5 shows how a solution for the UCP with N production units and with time horizon of T hours might be encoded. Binary values are used to represent the on/off status of each unit in each time period (1 = on, 0 = off). Since the whole solution is represented as one binary string the problem is no longer subdivided into units and time periods.

The minimal up and down constraint causes difficulties in GA, as it is hard to make sure that a solution is feasible with respect to them after one of the operators have been used. In most cases is it necessary to relax theses constraints and then either penaltielize

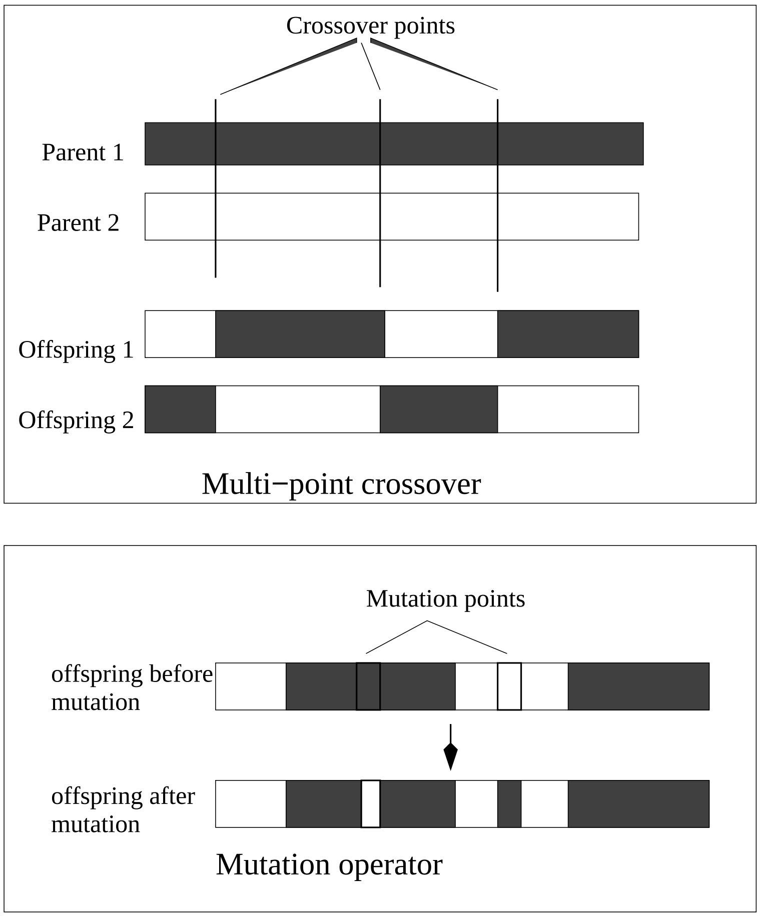

Figure 4.4: An example of the crossover operator (multi-point)and the mutation operator.

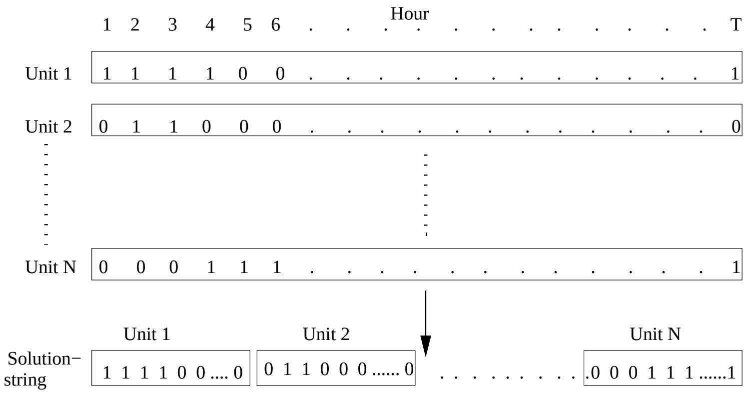

Figure 4.5: Binary representation of a UCP solution. 1 means that unit is active, 0 means that unit is un-active.

infeasible solutions in the objective function or repair them by some mechanism after they have been generated. The GA works well for the UCP in the sense that it provides flexibility in modelling different constraints. It is also quite good in locating good areas in the solution space but sometimes fails to locate good solutions within that area. To compensate for that hybrid methods such as combination of GA and SA [30] are sometimes used. More problem specific genetic operators than those discussed above can also increase the quality of the solution. GA can handle a relatively large problem size but on the expense of computation time which increases considerably with the size of the problem. Further description on the GA solving the UCP and GA in general can be found in [5, 7, 9, 17, 21, 23, 26, 28, 31, 33].

4.3.3 Local search strategies

Tabu search and simulated annealing can be classified as local search strategies. Local search consists in moving from one solution to another according to some definite rules [26]. A local search strategy starts from an arbitrary initial solution $x \in X$ where X is the finite set of all possible solutions. At each step n, a new solution x_{n+1} is chosen in the neighbourhood $N(x_n)$ of the current solution x_n. Neighbourhood $N(x)$ is a subset of X.

One way to select a solution in the neighbourhood of the current one is to search for all solutions in the neighbourhood and pick the best one x_{n+1} i.e. $f(x_{n+1}) \leq f(x), \forall x \in N(x_n)$. If the best solution in $N(x_n)$ is only picked if it is better than the current one is the method called Steepest Descent. An example of neighbourhood search can be seen in figure 4.6. If only 1 solution is checked in each neighbourhood and it is picked if it is better than the current one then is the approach called Greedy. It can take an extremely long time to search for best solution in the neighbourhood so it is often considered enough only to search in a limited number of points in the neighbourhood $N(x)$, noted $N'(x)$, and select the best one found. This is a combination of Steepest Descent and Greedy. Steepest Descent and Greedy have the disadvantage that they tend to get stuck in local minima. A local minima is a solution x_n with function value $f(x_n)$, where no $x_{n+1} \in N(x_n)$ can be found so that $f(x_{n+1}) < f(x) \quad \forall x \in N(x_n)$ and $f(x_n) > f^*(x)$ where $f^*(x)$ is the best solution in X or global optima.

To avoid these local minima, some mechanism have been implemented into the search such that it does sometimes approve worse solution and is therefore not completely descent. The SA and TS are examples of this and they will described in the following.

Simulated Annealing

Simulated annealing (SA) is a stochastic search method, which tries to simulate cooling process of a metal. Usually when using SA, one does not search in several points in $N(x)$ but 1 random solution is selected. In every iteration the algorithm approves this random solution if it gives a better value than the present one. It also approves a worse solution with some probability. This probability gets smaller as the difference between the current solution and the one under consideration gets larger. The probability also

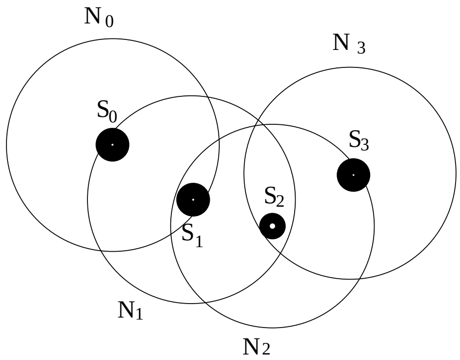

Figure 4.6: Search Space
Notation: S_0 : initial solution
S_i: best solution found in the neighbourhood. $N_{i-1}, i = 1, 2, ..., n$
N_i neighbourhood around $S_i, i = 0, 1, 2, ..., n-1$.

decreases with number of iterations taken, so in the end of the process hardly any of the worse solutions are approved. The probability of accepting a worse solution is usually given by equation 4.5, which is called the cooling function. There are actually more possibilities of a cooling function but they will not be represented here. Here Δc is the difference between the current solution and the solution under consideration. R is a uniformly distributed random number between 0 and 1. $T(i)$ is a control parameter in step i which starts large and then decreases with number of iteration according to some function. A classical schedule for decreasing $T(i)$ is illustrated in figure 4.7. Starting with temperature T_0, the temperature is maintained constant for L consecutive steps. Then the temperature is decreased by multiplying it by a constant factor α ($0 < \alpha < 1$, usually closer to 1) after each series of L steps [26] , hence after kL steps the temperature is as shown in equation 4.6. A worse solution is approved if the result from equation 4.5 is smaller than a uniformly distributed random number $R \in [0, 1]$. This property of accepting worse solutions with some probability helps the algorithm to escape local minima.

$$e^{\Delta c/T(i)} \tag{4.5}$$

$$T(kL) = \alpha^k T_0 \tag{4.6}$$

There are many possible criteria that can be used as a stopping rule. They are for example:

- Best objective value found f^* has not been improved by at least ϵ_1[5] % after K_1 consecutive series of L steps.
- Number of accepted moves is less than ϵ_2 % of total number of steps in L.

[5] ϵ is very small number

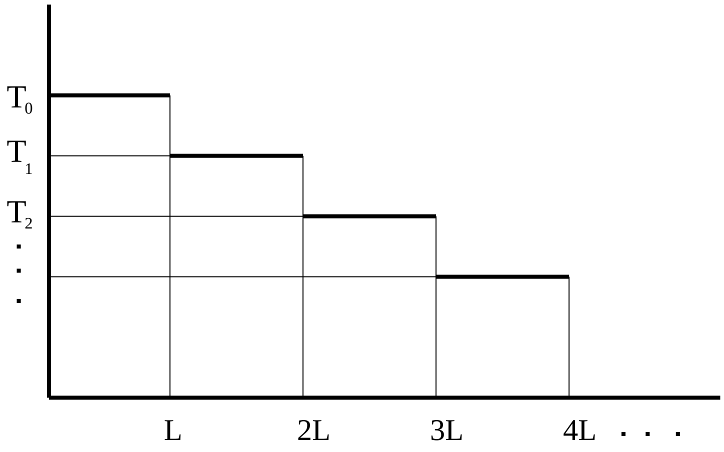

Figure 4.7: The most common temperature schedule

- Total number of iterations has exceeded some threshold T.

SA is a powerful technique to solve CO problems and is in many ways well suited for solving the UCP. It has the disadvantage of taking long computational time. The main advantages of SA are:

- It can find a high quality solution not strongly dependent on the initial solution.
- It does not need a complicated mathematical model of the problem under study
- It does not need a large computer memory
- It can be used on problems of almost any possible form [1]

An example of SA use to solve the UCP and SA in general can be found in [1, 26, 27, 28, 35, 37].

Tabu search

Tabu search is based on neighbourhood search with local optima avoidance, which in a deterministic way tries to model the human memory [28]. It searches for the best solution in some neighbourhood $N(x_n)$ of the current solution. Even if no improving solution x_{n+1} can be found in the neighbourhood than the current one x_n, the algorithm moves to the best found solution in $N(x_n)$. One possibility would be to search for the best solution in $N(x)$ but that is too time consuming so in most cases only a sub-neighbourhood $N'(x) \subset N(x)$ is searched, see figure 4.8.

There is a possibility that the best solution found in a sub-neighbourhood $N'(x_n)$ is x_{n+1} and simultaneously that x_n is the best solution in $N'(x_{n+1})$. This would result in cycling between these two solutions. To avoid this it is tried to implement memory

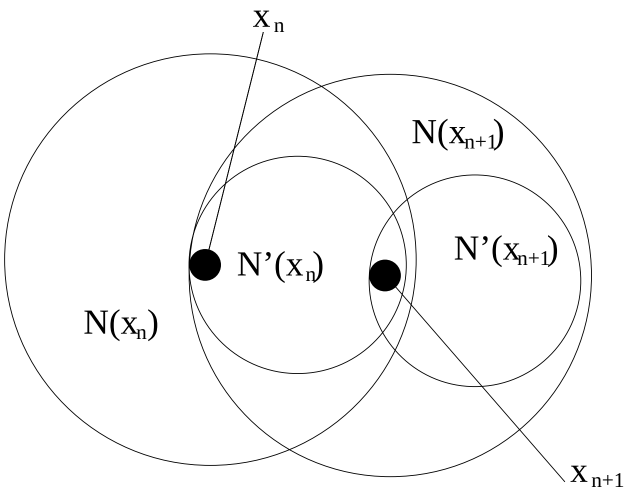

Figure 4.8: Example of neighbourhood $N(x)$, sub-neighbourhood $N'(x)$ and two current best solutions.

into the Tabu Search strategy. The memory is implemented by the implicit recording of previously visited solutions or previously made moves. These solutions (or moves) are kept in the "tabu list" which holds solutions that have been visited in the resent past and are "tabu" or forbidden. A new solution is then found in a sub-neighbourhood $N(x, TL)$ where TL indicates the tabu list. All solutions in $N'(x)$ are also in $N(x, TL)$ except for those in the TL. This memory raises a technical problem as every new found solution has to be compared to every solution in the tabu list. This results in much computational overhead, in comparing solutions. An alternative is to store some characteristics of the solution or simply forbid similar moves e.g. change in some unit for UCP.

The TL never holds more information than about a certain number of attributes defined by its tabu list length ($TL\ length$). When a new attribute is inserted to the list the oldest one in the list is deleted. There are two things that can lead to a worse solution than the current one is chosen. First if there is no better solution found in the neighbourhood secondly if all better solutions are rejected as they are in the tabu list. In practice is the best solution found so far kept, because there is no guaranty that the search will stop in the best solution found. Stopping criteria possible to use in TS can be:

- The total number of iterations taken has reached some threshold
- The number of iterations taken since best found solution was updated has reached some threshold.

Often good solutions are lost as they are forbidden by the TL. This arises a need for a aspiration criteria, when the aspiration criteria is met then a "tabu" solution is allowed. A example of aspiration criteria is:

- A solution rejected by the TL results in better objective value than the best solution found before in $N'(x_n)$
- All other solutions (moves) are rejected by the TL, then select the best solution found.

One disadvantage of the TS algorithm is the overhead involved in comparing solution to the ones in the tabu list. Another disadvantage is the overhead involved in copying solutions when searching for best one in a neighbourhood $N(x_n)$. The main advantages of TS are:

- It can find a high quality solution not strongly dependent on the initial solution.
- It does not need a complicated mathematical model of the problem under study
- It only has one parameter the tabu list size *TL length*.
- It can be used on problems of almost any possible form [1]

An example of TS use to solve the UCP and on TS in general can be found in [4, 15, 19, 20, 21, 22, 26, 28].

4.4 Conclusion

In the chapter most common methods used to solve the UC problem have been presented and described. They are sorted into three groups depending on how they solve the problem and the quality of the solution they find.

Exact methods can be well suited to solve the UCP, especially when the problem size is not too large and the objective function is linear. All exact methods have the property that they find optimal value. Three different exact methods where introduced they were extensive numeration, dynamic programming and branch and bound. Extensive numeration is the least practical as it requires calculation and keeping of every possible solution. It has extreme calculation time and requires much memory. The other two methods try to limit the solution space. The branch and bound method is available in GAMS. It was therefore straight forward to implement the UCP in GAMS to solve the problem.Optimal could be found for smaller problem sizes. In cases were an optimal solution could be found it was used to evaluate the quality of the solution found with TS. The Gams code can be found in appendix **??**

The B&B method has the problem that the decision tree can get extremely large and optimal solution can not be found. In those cases GAMS returns the best found solution and the relevant duality cap. In [13] was a very similar UCP with CHP system solved using B&B in GAMS so it is not of much interest to do that again. 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, and if the size of the tree becomes more extreme in size it gets too large for the computer memory. It was decided to not solely to use B&B approach to solve the UCP both because of it limitation on problem type and size and because it has been tried before for almost the same problem.

Two relaxation methods were introduced they are, Lagrangian and linear relaxation. Lagrangian relaxation (LR) has often been used in the literature to solve the UCP. LR often has the problem of finding good multipliers and therefore some heuristic methods are often used to find them. This is especially the case for large UCP. The implementation of LR is rather complicated and there was no experience by the author in that kind of implementation. It was decided not to use LR because of its problem of finding multipliers and difficulty to implement. Linear relaxation can find solution with similar objective value as the non-relaxed one but suffers from the fact that it often produces infeasible solutions. Secondly there is no guaranty that the solution structure will be anything like the optimal one because both start up costs and minimal up and down times are ignored. Linear relaxation not solve the UCP actually, it solves a large EDP where minimum capacity in all unit is zero. Linear relaxation has been used in the *Balmorel* solving a larger version of a similar problem, so it was not of interest here.

Metaheuristics methods have the property that they can be used on problems of various types and forms. They can produce near optimal solution and factors like size of problem, type of objective function and constraints are not crucial factors on whether the method can be used or not. The heuristic method priority queue can possibly be an approvable method but the quality of the solution can be bad and the algorithm always converges to same solution. GA can be good approach to solve CO problems and it is especially good in finding good solution areas. The minimal up and down constraints causes problems and almost the only possibility is to relax them. Then either put penalty on them in the objective function or a infeasible solution can be repaired by some approach. It is also not as transparent to see what is happening in each iteration step as in the local search methods TS and SA.

TS and SA are as mentioned before local search algorithms, which in a deterministic way try to avoid local minima. It is easier to make sure that solutions created are feasible with respect to minimal up and down constraints for these algorithms than GA. These methods are easier to implement than GA. It is also easier to follow what is happening in a local search strategy as it is known that the solution structure does not change much between iterations. The choice was therefore between either TS or SA and there was no reason to choose one over the other. It was decided to implement a TS algorithm. The reason for choosing that one rather than SA is mainly because it seemed interesting to implement its neighbourhood search and tabu list and see how it would work.

Chapter 5

The Solution Method

This chapter describes the algorithm implemented to solve the combinatorial part of the UCP as the problem was formulated in chapter 2. The EDP problem is solved with a solver that was discussed in chapter 3. The EDP is an optimization problem of heat and electricity were the production is connected in CHP plants. In last chapter few methods possible to solve the UCP were discussed. It was found most interesting to use the Tabu Search (TS) strategy to solve the problem. In this chapter the TS algorithm that was implemented will be described and other factors related to TS. The chapter is divided into two main sections, first the implementation of the neighbourhood search and secondly the tabu list implementation. The description of the neighbourhood search is divided into two parts, first is the general idea discussed and then a more detailed description is given. The generation of an initial solution, the stopping criteria of TS and the EDP calculation is also discussed but not as much emphasis is put on that. In section 4.3.3 the TS strategy was described and the description here is based on the description in that section. Same notation is used here as in section 4.3.3.

5.1 The neighbourhood search

In neighbourhood search algorithm like TS, it is very important to search the neighbourhood as widely as possible to be able to find a good feasible solution. That requires a good search strategy. When designing a neighbourhood search strategy, there are three main things that have to be kept in mind, they are:

- The way new solutions are generated.
- The size of the neighbourhood.
- Should the neighbourhood search only generate feasible solutions or both feasible and unfeasible solutions.

These three points actually all are interconnected. The way a new solution is generated influences the size of the neighbourhood. If unfeasible solution space is searched then the neighbourhood is larger. The way new solutions are generated controls whether the infeasible solution space is also searched. The first point is of most importance, and here will follow description on how a new solution is generated. The other two points will be mentioned in connection with that discussion.

5.1.1 Generating a new solution

When generating a new solution the hardest constraint is the minimal up and down constraint. One possibility would be to relax the minimal up and down constraint. A description on such an approach can be seen in section 5.1.3. It was decided not to relax these constraints, but rather search only in the feasible solution space. Many changes are allowed on a solution in one step, which indicates a large neighbourhood. The neighbourhood search used here is inspired from [1] and [19], they actually both describe the same neighbourhood search. A few modifications have been made on the search to improve it. The general idea of the neighbourhood search is that a change is made on the opening status from ON to OFF or vice versa of one unit over some time periods of same unit status[1], a discussion about neighbourhood size can be found in 5.1.2. The pseudo code, which describes how the search works is shown here below. A few other neighbourhood search methods were tried which all work on the same principles but the one here below is the most complicated one and it worked best. The result from the tests are shown in the next chapter about computational results.

The pseudo code for the neighbourhood search is as follows:

Notation:

U: uniform distribution
uc: *unit commitment solution matrix*
Time: *the set of time periods where it is possible to change unit status over a continues string of same unit status, e.g. see the small arrows at the bottom on figure 5.2, 5.3 or at top of 5.4.*
Length(Time): *for each Time, Length(Time) is the set of number of time periods it is possible to change unit status, see figure 5.4.*

0. *Begin*
1. *Select unit* $\sim U(1...N)$ *and time* $\sim U(1...T)$
2. *if* $uc(unit, time) = OFF$
3.   *Find for how long unit is OFF*
4.   *Find Time for the unit and time*
5.   *Find Length for each element in Time*

[1]unit status refers to unit being active or inactive.

```
        Select random Time → time and an element out of Length(Time) → L
        Turn unit on from time to time + L
        if( spinning reserve infeasible )
                go to line 1
end
```

The case $uc(unit, time) = ON$ is proceed in a similar manner.

Examples of 3 different modifications are given in figures 5.1, 5.2 and 5.3. The small arrows point at time periods were it is possible to open the unit. How long it can be opened depends on the time period selected. Feasibility of the solution matrix is always preserved. The unit can be opened from one to $tOff$ time periods. It should be noted that the units are opened from left to right on the figures. If last time period is selected then the unit is opened from the left most position such that it is opened to the end of the closed period.

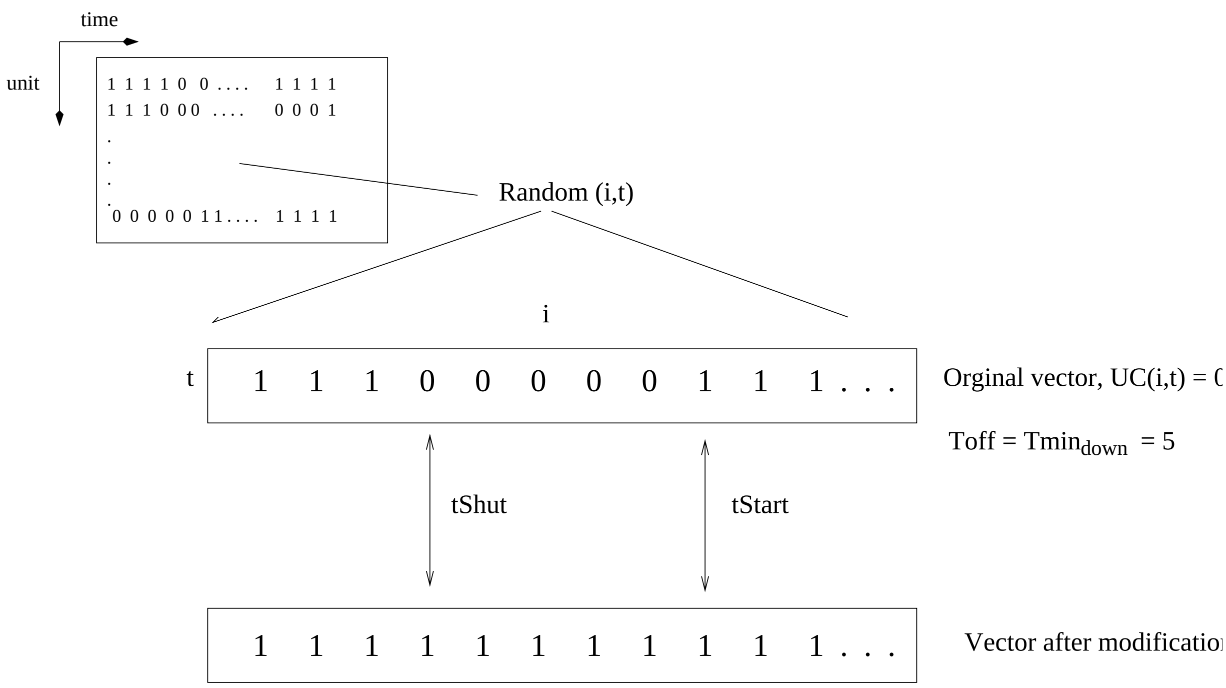

Figure 5.1: An example of how the neighbourhood search algorithm works. Here $Tmin_{down}$ is equal to $tOff$.

More detailed description

In principle the algorithm works as follows. First a unit and a time period are selected at random from the UC solution matrix. If unit is OFF find out when the unit was turned OFF called $tShut$ and when it is turned ON again, called $tStart$ (if unit is ON

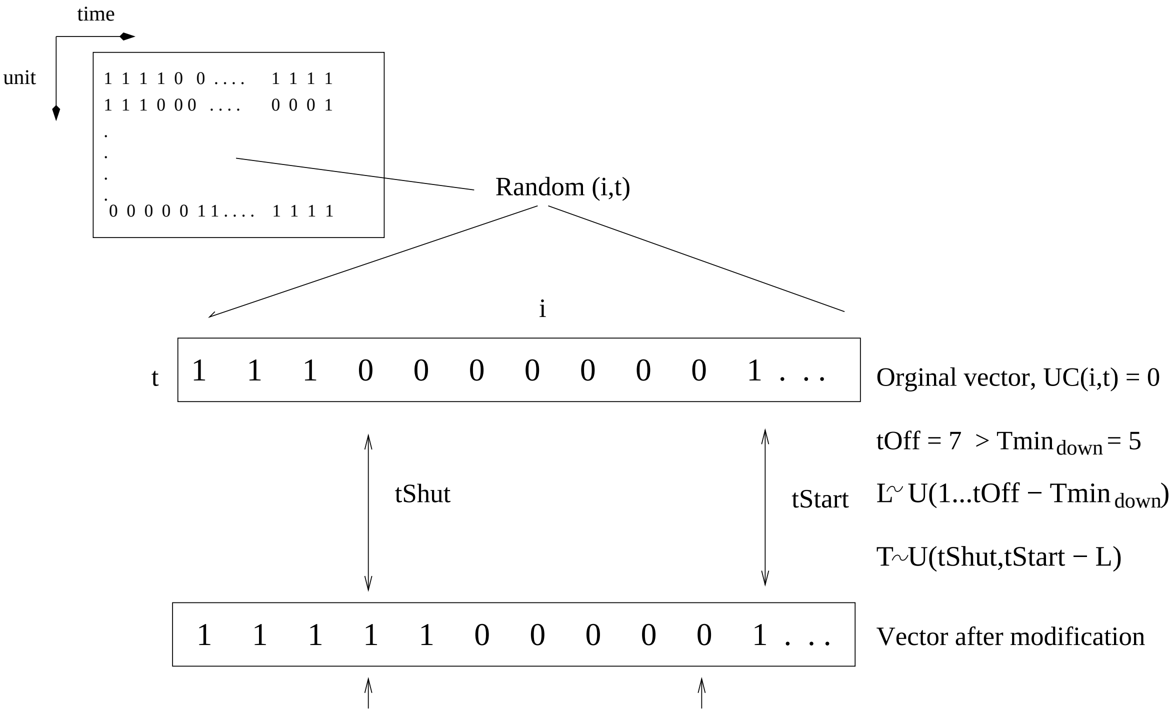

Figure 5.2: An example of how the neighbourhood search algorithm works. Here $tOff$ is lager than $Tmin_{down}$ and less than 3 $Tmin_{down}$. The small arrows at the bottom of the figure point at places were it is possible to turn unit ON.

the procedure is similar). Then calculate for how long the unit is OFF called $tOff$. There are three possible scenarios that can occur depending on the length of $tOff$.

1. $tOff$ is equal to $Tmin_{down}$. In this case the unit is turned ON for $Tmin_{down}$ time periods starting from time $tShut$. An example can be seen in figure 5.1.

2. $tOff$ is larger than $Tmin_{down}$ and less than 3 $Tmin_{down}$. In this case it is possible to open unit from both ends but not in the middle, small arrows in figure 5.2 point at time periods were the unit can be turned ON. A number L is selected at random from 1 to $tOff$ - $Tmin_{down}$ and $time$ as either $tShut$ or $tShut - L$. If the selected L is $L = tOff$ - $Tmin_{down}$ then it is rewritten as $tOff$. Then unit is turned ON from $time$ to $time + L - 1$.

3. $tOff$ is larger or equal to 3 $Tmin_{down}$. In this case it is also possible to open unit in the middle of the closed period. An example of possible time periods is shown with small arrows on figure 5.3, and in figure 5.4. The time periods to select from are in practice $tShut$, $tStart$ and all time periods from $tShut + Tmin_{down}$ to $tStart - 2Tmin_{down}$. Then a $time$ is selected at random from those time periods. If a period is selected from $tShut + Tmin_{down}$ to $tStart$ - $2Tmin_{down}$ then L is set

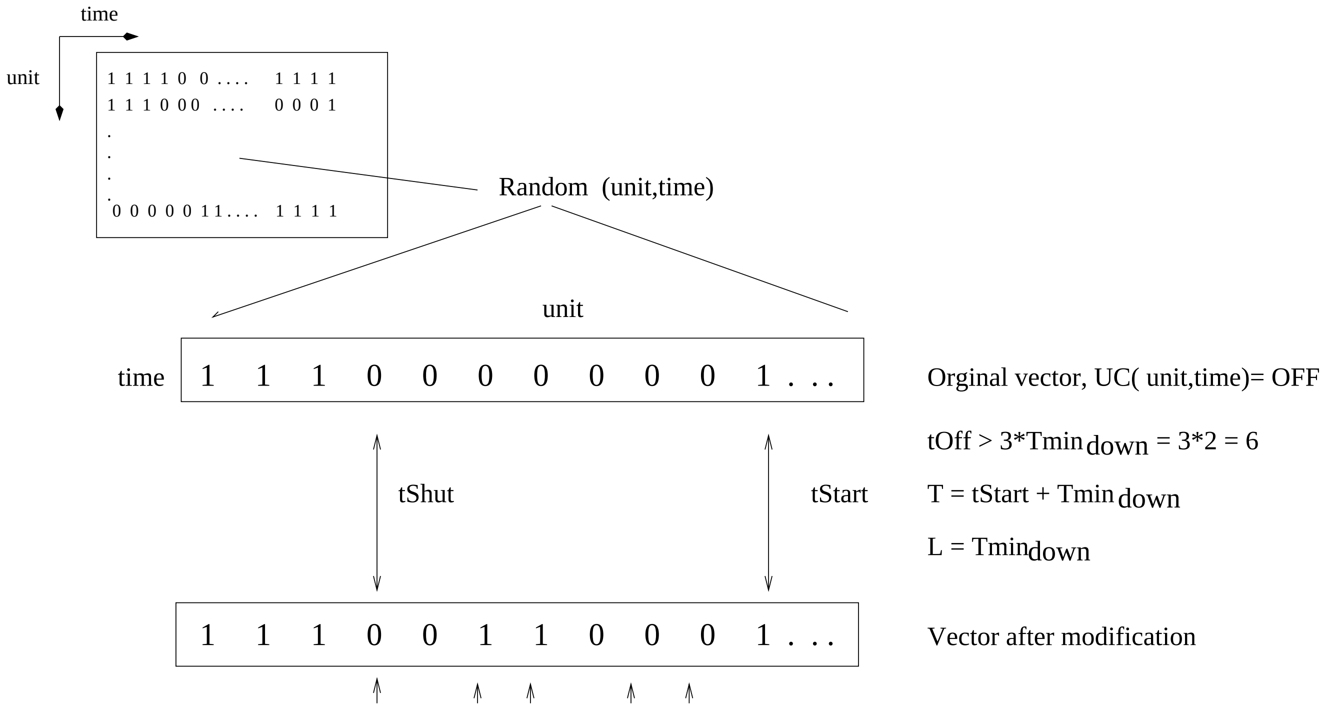

Figure 5.3: An example of how the neighbourhood search algorithm works. Here $tOff$ is larger than 3 $Tmin_{down}$. The small arrows at the bottom of the figure point at places were it is possible to turn unit ON.

to $Tmin_{down}$[2]. Otherwise L is selected at random from 1 to $tOff$ - $Tmin_{down}$ and if $time$ is $tStart$ then is the $time$ rewritten as $time = tStart - L$. Finally the unit is turned ON from $time$ to $time + L - 1$.

After the modification it is verified that the solution is feasible with respect to spinning reserve constraint. If a solution fulfils spinning reserve constraint then the solution has capacity to fulfil demand. If spinning reserve is fulfilled then a new trial solution has been found. If not a new solution is created using the same UC solution matrix as last time. For more detailed description of the neighbourhood algorithm another pseudo code for the neighbourhood search is given. All C++ implementation of the TS algorithm can be found in appendix **??**

[2]it would actually be possible to open unit for time periods up to $tStart - Tmin_{down}$ but that is not done here

Notation:
i: index of unit.
t: index for time.
uc: The unit commitment solution matrix
$tShut$: time where unit is turned off.
$tStart$: time where unit is turned on.
$tOff$: the total time unit is off.
$Tmin_{down}$: minimal off time for unit i.
T: total number of time periods.
N: number of production units.
L: an integer.
$U(1...N)$: a uniformly distributed number from 1 to N.
$U(1, N)$: randomly select either 1 or N.

1.Select $unit \sim U(1...N)$ and $time \sim U(1...T)$
2.if $uc(unit, time) = OFF$
3. Find the time where unit was turned OFF $tShut$.
4. Find the time where unit is turned ON $tStart$.
5. $tOff = tStart - tShut$
6. if (tOff $= Tmin_{down}$)
7. L $= Tmin_{down}$
8. $time = tShut$
9. if(tOff $> Tmin_{down}$ & tOff $< 3Tmin_{down}$)
10. $L \sim U(1...tOff - Tmin_{down})$
11. $time \sim U(tShut, tStart - L)$
12. if ($L = tOff - Tmin_{down}$)
13. $L = tOff$
14. if(tOff $\geq 3Tmin_{down}$)
15. $time \sim U(tShut, tShut{+}Tmin_{down}...tStart{-}2Tmin_{down}, tStart{-}Tmin_{down})$
16. if($time = tShut$)
17. $L \sim U(1...tOff - Tmin_{down})$
18. else
19. if($time = tStart - Tmin_{down}$)
20. $L \sim U(1...tOff - Tmin_{down})$ and $time = tStart - L$
21. else
22. $L = Tmin_{down}$
23. Turn unit ON for time periods $time$ to $time + k, k = 0, 1, ..., L - 1$.
24. if(spinning reserve feasible)
25. Stop algorithm
26. else
27. Go to start of algorithm

The case $uc(unit, time) = ON$ is proceeded in similar manner.

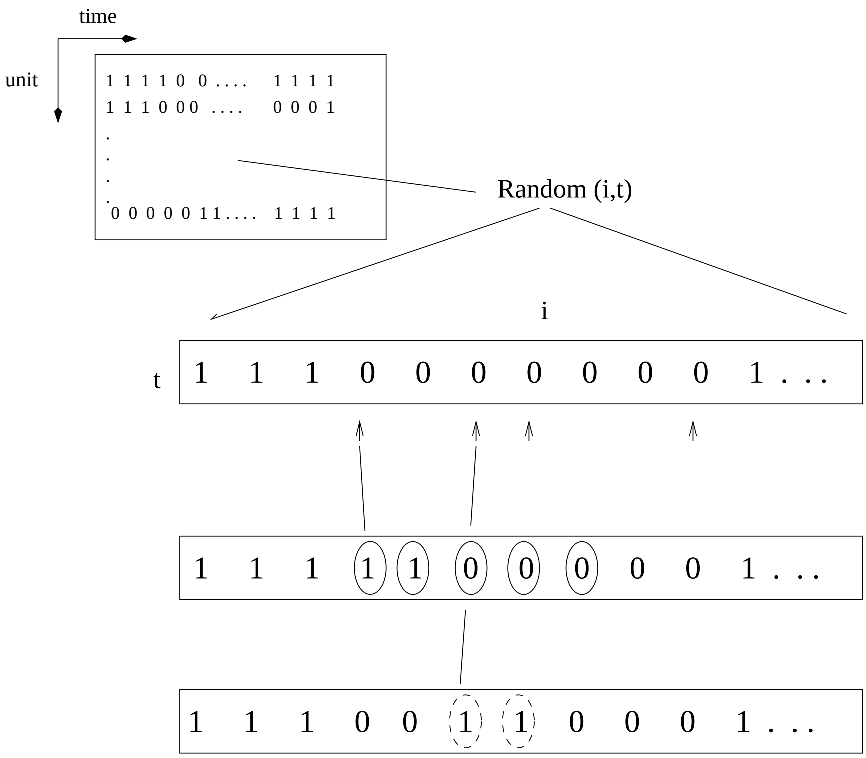

Figure 5.4: An example of how the neighbourhood algorithm works. Here is $tOff > 3\ \ Tmin_{down} = 3 * 2$. The arrows point at places where it is possible to turn unit ON without breaking minimal up and down constraints. If the time period at the beginning is selected (shown in the middle of the figure) then it is possible to open the unit from 1 up to $tOff - Tmin_{down}$ time periods (shown with ovals). If one of the time periods in the middle is selected (show at the bottom of the figure) then it is only allowed to open unit for $Tmin_{down}$ time periods (shown with broken line ovals).

5.1.2 Properties of neighbourhood search

It is common that the neighbourhood is defined as small as possible. Here the size of the neighbourhood actually is rather large. The neighbourhood of the current solution is defined as: the set of all solutions that result from a change in the opening status from ON to OFF or vice versa of one unit over some time periods of same unit status. This indicates that extremely many solutions are in the neighbourhood of the current one. For example it is allowed to open unit for time period up to $tOff$, in one iteration, and $tOff$ can be as large as T. The reason for having such a large neighbourhood is to have much variability between solutions in the neighbourhood. It was found out that all good solutions to the UCP have the form that a few units are ON at almost all time periods. By allowing such a large change the possibility of gaining such a solution increases. The neighbourhood search in [19] only allowed opening (or closing) unit from the start of the closed (or open) period. That limits the search considerably and for example it is extremely hard to gain solutions where a unit is only open for a short time period in middle of the whole time period T. Most good solutions to the UCP have the form that some units are only ON for few hours while the demand is highest, in so-called peak units. It was therefore allowed to open/close unit in the middle of the continuous string of closed/opened unit. In the implementation here it is also allowed to change unit status from the end of a continuous string of same unit status.

It is not practical to search the whole neighbourhood for a candidate solution. If all solutions in the neighbourhood were checked then the algorithm would always converge to the same solution for same staring solution. It is also extremely time consuming to search the whole neighbourhood. Therefore only a limited number of solutions in the neighbourhood are checked, see figure 4.8. It is not obvious how many the candidate solutions should be and that is a parameter that have to be tuned for every problem. After the neighbourhood has been searched the algorithm moves to the best solution found in the neighbourhood.

5.1.3 Relaxing minimal up and down constraint

The neighbourhood is defined such that it only contains feasible solutions. All solutions generated (visited) are feasible with respect to all constraints. The minimal up and down constraints are the ones that limit the searching space most. It is possible to define the neighbourhood such that also holds infeasible solutions. Such a neighbourhood would then require that infeasible solutions were either repaired or that they are penaltielized in the objective function to make them unattractive. To repair solution would though sound more reasonable as in that case it would not be necessary to calculate the objective value for an unfeasible solution.

An example of infeasible solution that is repaired can be seen in figure 5.5. If same principle is used to only change the unit status in a continuous string of same unit status, it would result in that the length of three continuous strings of same unit status had to be calculated. The repair would then result in turning a unit ON or OFF for some number of time periods to hold minimum up down constraints. That kind of approach would not necessary be more complicated to implement. Usually one also ends up with a solution that can be obtained with the neighbourhood search as it is.

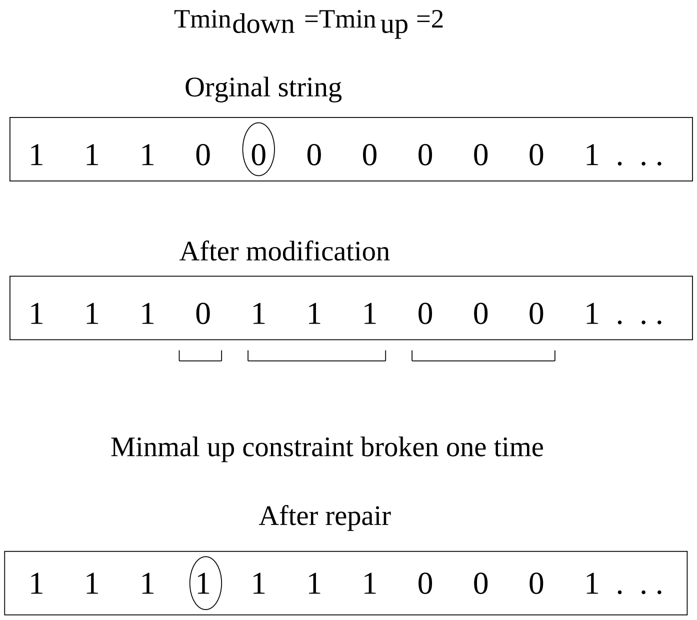

Figure 5.5: An example of how a infeasible solution could be created and repaired. First is a unit and time period selected at random (see oval at top string). Then is the unit turned ON for few hours see middle of figure. Then do the length of the tree segments have to be calculated (showed in middle of figure). Here the solution is infeasible then it is repaired. The solution is repaired by turning unit ON at time period shown by a oval at bottom of the figure.

5.2 The tabu list implementation and related factors

Tabu search has (as the name indicates) the tabu list (TL). The purpose of the TL is to avoid cycling and local minima. There are many possibilities to implement a TL for the UCP. One possibility would be to store the whole T x N (time periods x units) UC solution matrix for recently visited solutions. That would lead to extremely many comparisons. It is therefore more practical to store some characteristic of the solution or move that would lead to a recently visited solution. An example of move could be a change of unit status for a certain unit. Here it was decided to store some characteristic of the solution in the TL. Each entry of the TL holds a vector of length N. This vector holds information on the number of times that each unit has been ON. This approach of tabu list does actually give the possibility that a solution is rejected that has not been visited before, but that is often the case with Tabu Search. The Tabu Search algorithm was implemented such that the number of solutions in the sub-neighbourhood is always equal to the number of solutions in $N(x, TL)$, or $|N(x, TL)| = |N'(x)|$. Every newly created solution is compared to the TL as soon as it is created and if it is rejected by the TL then a new solution is created immediately so it is made sure that $|N(x, TL)| = |N'(x)|$.

It was also tried to use another TL where each entry would hold a vector which keeps the number of units that are ON in each time period, this TL will be referred to as the *time TL*. The idea was that the *time TL* would only be checked if a solution had been declared "tabu" by the other TL. If the solution would also be declared "tabu" by the *time TL*, then this solution would be declared forbidden and a new candidate solution found. The idea was to reduce the possibility that a solution was rejected that not ad been visited. It worked better to not use the *time TL* but the results from the test will be shown in chapter 6.

The only parameter in the Tabu Search algorithm is the tabu list length (except the number of solutions checked in each neighbourhood). Theoretically the results are dependent on the length of the tabu list. Long tabu list should help the algorithm to escape local minima while too long tabu list prevents effective exploration of promising areas by hindering the choice of neighbours for intensification purposes [15]. Various tabu list lengths were tested and compared.

The pseudo code for the Tabu search algorithm is as follows:

Notation:
x_n^*: Best solution found in neighbourhood of solution x_n.
x^*: Best solution found in solution space, X.
n: number of solutions checked in total.
j: number of solutions checked since x^* was updated last.

Begin
    Generate initial solution
    while stopping criteria not met
        Generate $N(x_n, TL) \subset N(x_n)$
        Where $|N(x_n, TL)| = |N'(x_n)|$
        $x_n^* = \min(N(x_n, TL))$
        if $f(x_n^*) < f(x^*)$
            $x^* = x_n^*$ and j = 1
        n = n + $|N(x_n, TL)|$ and j = j + $|N(x_n, TL)|$
    end while
end

5.2.1 Initial solution and stopping criteria

In principle a neighbourhood strategy like TS should not be very depended on the initial solution, but in practice it often is. The initial solution must be feasible with respect to all constraints and it has to be possible to iterate to a good solution from it. Three different initial solution generators were tested and the results are represented in next chapter. The one that gave best results was surprisingly the simplest one. In that case the start solution is always the same, with all units ON at all hours.

Two stopping criteria were used simultaneously and they are:

- if the number of iterations taken in total had reached some threshold called n_{max}
- if the number of iterations since the best found solution was updated called j_{max}.

The size of j_{max} was set around 15% of n_{max} and in over 90% of the runs the algorithm did stop because j_{max} was reached. It was tested if better solutions could be found if j_{max} was increased considerably, but that did not affect the solution quality significantly. The results are shown in next chapter.

As noted in section 4.3.3, is the best solution found kept in TS. That was done in the implementation described here. Contradictory to SA that theoretically always ends in converging to a local minima, TS just continues searching and might move very far from best found solution. Therefore is it good to keep best found solution and stop the algorithm when it ha not be updated for some time or tat some maximum number of iteration have been reached.

5.3 The EDP

For every solution found by the Tabu Search algorithm the EDP has to be solved. In that sense can the EDP be referred to as the objective function for the UCP. The EDP is solved with the heuristic that was descried in chapter 3.

As the neighbourhood search is implemented here do the UC solution matrix only changes for one unit over few time periods at each iteration. On average the change is 6 time periods per iteration. In the EDP heuristic start-up cost and minimal up and down constraints do not apply which indicates that time periods are independent and are therefore solved individually. This gave the possibility to update the objective function instead of solving the EDP problem for the whole UC solution matrix in each iteration. The production cost in each time period is kept. When a new solution is found the production cost is only calculated for the time periods where the unit status was changed. Then the objective function value is obtained by summing the producing cost in each time period. This leads to less overhead related to calculating the objective function. The calculation time did decrease but the difference was not big. The reason can be that too much calculation time is involved in copying UC solution matrices and comparing solutions to the TL.

5.4 Technical improvements

Many improvements have been done to increase the quality of the final solution and to decrease the execution time, though are there is still possible to improve this algorithm further. This section gives few ideas how the performance of the algorithm could be improved.

The TL is implemented so it stores the number of times that each unit is ON. That is of course not the only way to make the TL. In the literature the TL often simply forbids some moves. For the UCP a move can be defined as a change in the opening status of one unit. It was not tested to have such kind of TL. Based on the experience of the problem it is not expected to give better solutions. It is though likely that it would be possible to have shorter TL because for example if $TL\ length = 5$ then 20 % of all moves are forbidden when the number of units are 25.

The neighbourhood search in TS involves many copying of UC solution matrices. While searching in the neighbourhood every time a better solution is found than before the solution matrix has to be copied. The solution matrix must also be copied when the search in $N(x, TL)$ is finished. Finally solution has to be copied when new best solution found in X is updated. This results in considerable computational overhead. At average does the UC solution matrix have to be copied 5 times for each neighbourhood search where number of search solutions in each neighbourhood is 35. That results in considerable computational overhead. This overhead could be reduced by only copying the part of the solution that is different from the current solution x_n. This would speed up computational time of the algorithm.

The neighbourhood search strategy gives the possibility to change the unit status in the middle of a string of continuous unit status. It only is allowed to change the unit status for time period equal to $Tmin_{down}$ (or $Tmin_{up}$), while it would actually be possible to change the unit status for time periods up to $tOff - Tmin_{down}$. For example in figure 5.4 it would mean that the unit could be opened for 2 or 3 time periods.

The TL length here is constant as is normally the case with TS. [2] introduced a concept called reactive tabu list. That involves that the length of the TL changes during the iteration process. There are few possibilities to implement this, one idea is that it get shorter when few solutions are rejected by the TL to increase converge rate. When the TL rejects many solutions is the length increased to help stepping out a minimum. It would be interesting to see if that could give better results.

The idea about aspiration criteria was discussed in section 4.3.3. It could be interesting to see that work on this problem. Especially the one about approving solution rejected by the TL if it results in better objective value than found before. If either aspiration criteria or dynamic TL should be implemented more interest would be in implementing aspiration criteria. The reason for that is cause it seems to be a tenancy in the TS to have problem locating the minimum in a "good" neighbourhood.

5.5 Conclusion

In this chapter the TS implementation for the combinatorial part of the UCP has been described. In this chapter most emphasis was put on the general idea behind the neighbourhood search and the tabu list implementation. Other things like initial solution creator and stopping criteria are of less importance. The updating of the "objective function" (EDP) has also been discussed, and how that is limited.

The neighbourhood search was implemented such that it only searches in the feasible solution space. The neighbourhood size was rather large. It was done to have much variability between solutions within each neighbourhood and to make it easy to gain many solution from the current one.

The TL, which is implemented here, is of a kind called "recency". Recency tries to simulate a $short-term$ memory. There is also possible to implement a form of $long-term$ memory, by the use of frequency measures. An example of this can be seen in [10] and [11]. This could possibly prevent that the same solution neighbourhood is visited again and again. It could be interesting as a future work for this algorithm to implement a $long-term$ memory.

Usually when solving the UCP the TL does not contain actual recently visited solutions. There are two main reasons for that. First that would often take considerable memory space and would involve extremely many comparisons, especially for the UCP. Secondly it is often interesting to not only forbid actually previously visited solution but also some solutions in the neighbourhood of the previously visited solutions. Here the TL was implemented so it kept the number of times that each unit had been ON. More ideas on TL for the UCP can be seen in [19].

In next chapter computational results of the TS algorithm are reported.

Chapter 6

Computational results

In last chapter was the TS algorithm, which was implemented to solve the combinatorial part of the UCP described. In this chapter its performance will be presented. When implementing a heuristic strategy it is very important to measure its performance. Most heuristic algorithms are stochastic which indicates that a variance can be expected in the solutions that the algorithm return. The final solution that the algorithm returns can be very different. The average performance is therefore of more interest then the best objective value found.

Tabu search itself has only one parameter that is the tabu list length. The number of solutions checked in each neighbourhood can also be considered as a parameter. In heuristics it is very important to tune all the parameters to get best performance. Here the tabu list length and the number solutions checked in each neighbourhood were tuned simultaneously. It also was tested to solve larger problems with time horizon up to 96 hours (4 days). The number of units is also increased up to 34 by splitting the larger units. The final section of this chapter is dedicated to comparison of TS and SA algorithm implemented by a fellow student working on the same problem.

Now there is time to discuss the problem at hand in more detail. The production units in the problem are located on the eastern part of Denmark [1]. In this area there are 25 units, which are subject to unit commitment, the production capacity of each unit type is shown in table 6.1. Considerable portion of the total capacity is in CHP plants which indicates how important it is to have a good solver for the EDP that can find good production in the CHP plants.

6.1 TS specific parameters and other related factors

In this section it will be described how all parameters were tuned and selected. There will also be a discussion about the initial solution creator selected, the neighbourhood

[1] Part of Denmark east of Great Belt

Table 6.1: Production capacity in each unit type for both products and areas.

		Extraction	Back pressure	Heat only	Condensing
	Electricity	1947	1060	0	1844
Capacity	Heat Urban	1519	1212	3400	0
	Heat Rural	640	1070	160	0

search strategy, the number of iterations necessary to gain good solutions and the form of the tabu list. In the tests in this section the time horizon used in these tests is 24 hours and each test is composed of 10 different runs of the TS algorithm.

6.1.1 Tuning parameters

For a heuristic strategy it is very important to tune all parameters such that the strategy gives best results. Here the parameters to tune are the TL length and number of solutions in the neighbourhood noted $|N'(x)|$. Both parameters were tuned simultaneously which means that all combination of TL length and $|N'(x)|$ were tested. TL lengths tested were 5, 10, 15, 20, 25 and 30 while $|N'(x)|$ was 1, 5, 10, 15, 20 and 25. Table 6.2 shows the order of the combinations in different test.

Table 6.2: The combination in each test.

Test #	1	2	3	4	5	6	7	8	....	36
TL length	5	5	5	5	5	5	10	10	....	30
$\lvert N'(x)\rvert$	1	5	10	15	20	25	5	10	....	25

The case $|N'(x)| = 1$ is actually very impractical. As mentioned before TS always picks the best solution in the neighbourhood regardless of whether that solution is better or worse than the present one. When $|N'(x)| = 1$ all solutions are approved and the final solution is some sort of average solution that is good for nothing else than possibly to measure the data structure. Having $|N'(x)| = 1$ is also extremely time consuming because for every new found solution the UC solution matrix first has to be copied 2 or 3 times. That is first as best found solution until now in the neighbourhood, then the same solution is copied as best solution found in $N'(x)$, finally if the solution is the best solution found in X then it has to be copied one more time.

The solutions from these tests are shown in figure 6.1, here the solutions are represented with box plots. A description on box plots can be seen in appendix **??**. It seems to be a tendency in the strategy to give better solutions when TL length and $|N'(x)|$ are large. The range is small at same time that the average objective value is rather low. This is

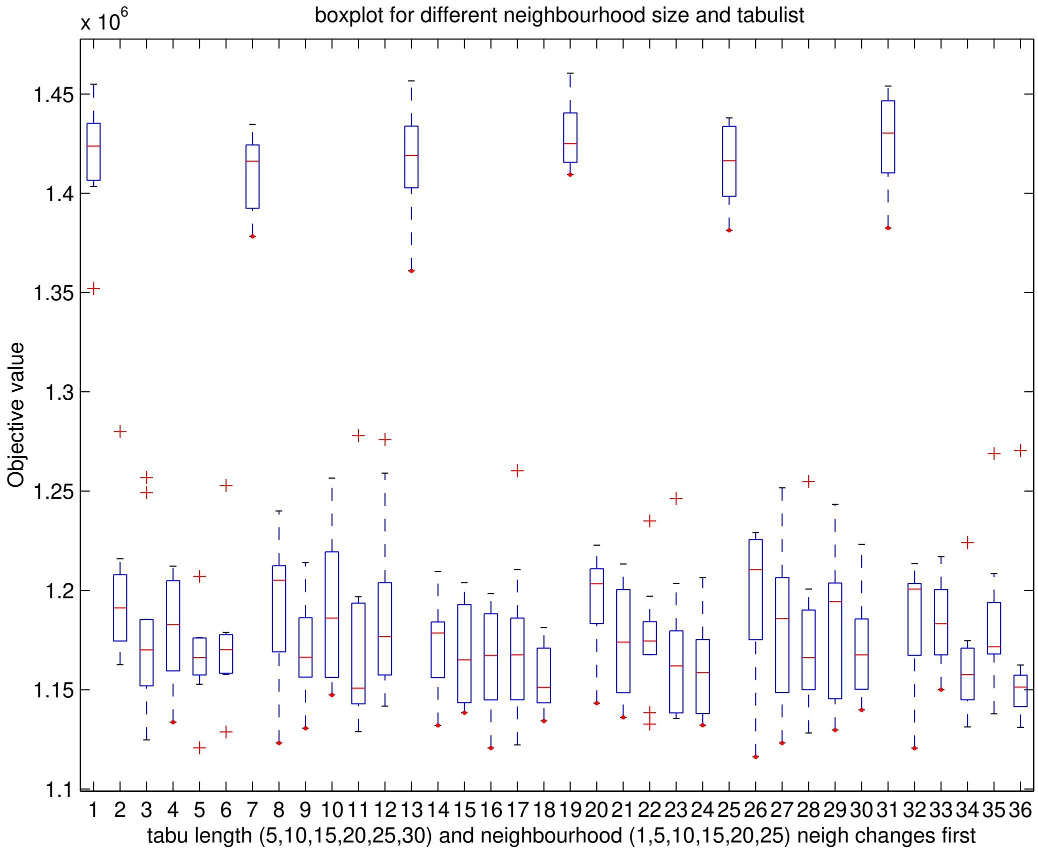

Figure 6.1: Box plot of objective value for different TL length and $|N'(x)|$.

especially the case in runs 34 and 36, in these runs the TL length is at its maximum and $|N'(x)|$ also at run 36. It therefore was tested to run the algorithm for larger values of the parameters. The TL length was 35, 40 and 45 while $|N'(x)|$ was 30, 35 and 40. The order of the tests are same as in last figure, that is the first TL length is tested with all $|N'(x)|$ before testing next TL length. The results can be seen in figure 6.2. Here the combinations 6 to 9 give best results and combination 8 is considerably better than the others. The average values of all tests are also lower than for those in figure 6.1. There is no reason for why this combination should give that much better solution than the others and most likely is it just coincident, then it was chosen to be used in the rest of the tests. This combination had TL length = 45 and $|N'(x)| = 35$. The test was run the second test and then did combination 8 give similar results as combinations 6 to 9. It was also tried to have longer TL length up to 60 and $|N'(x)|$ up to 50 but the results did not change significantly by that.

For the cases that were tested to tune the algorithm the execution time was also measured. Results from time measurements for the same test as in figure 6.1, can be seen in figure 6.3. It is interesting to see that there is a significant decrease in execution time as the $|N'(x)|$ increases. Smaller $|N'(x)|$ requires more solution matrix copying in total and therefore more time overhead is involved in that. Runs with larger $|N'(x)|$ also

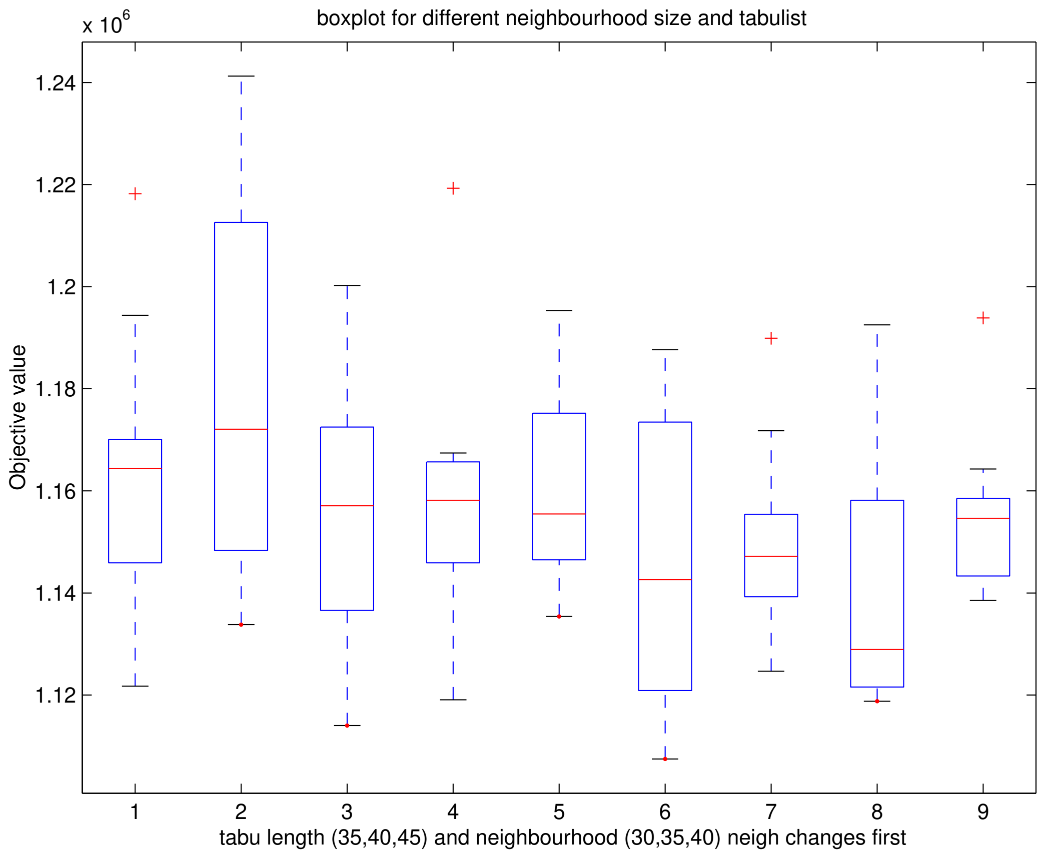

Figure 6.2: Box plot of objective value for different TL length and $|N'(x)|$.

converge faster to a minima because better solution is found in each neighbourhood.

6.1.2 Initial solution creator

The idea with meta-heuristic strategy like TS is to be able to find good solutions not entirely depended on the initial solution. That is in some sense true as the strategy finds different best solutions from same initial solution. But of cause some initial solutions do give better solutions than others.

To test the algorithm sensitivity to initial solution and also to find a good initial solution creator three different initial solution creators were tested and compared.
The three different initial solution creators are:

1. Start with an UC solution matrix where all units are closed at all houres and then changes are made on the UC solution matrix in same manner as in the neighbourhood search strategy until the solution is feasible with respect to spinning reserve constraint.

2. Simply start from a solution where all units are active at all hours. May sound

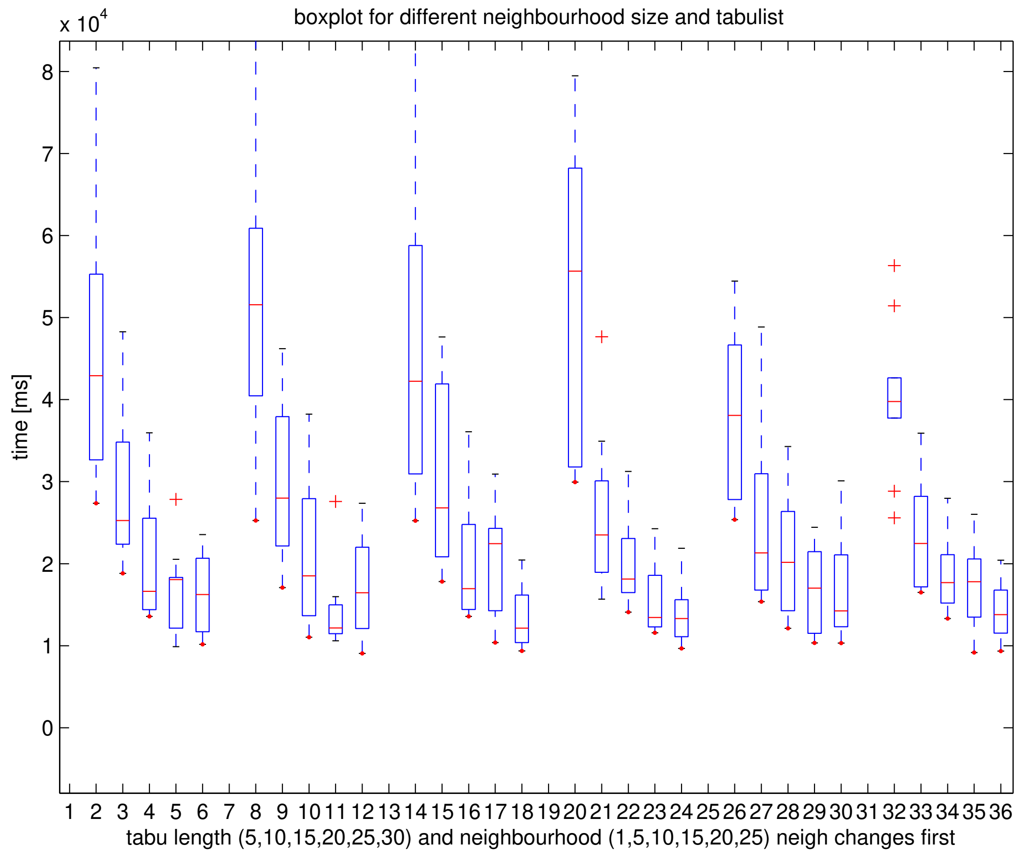

Figure 6.3: Box plot of computational time for different TL length and $|N'(x)|$. Note that the case $|N'(x)| = 1$ has been cut out.

strange to always start from the same solution but from this solution all other solutions can be reached easily.

3. Start with a solution where all units are active at all hours, then the neighbourhood search is called 10 times and feasibility preserved at all times.

In figure 6.4 the results from different initial solution creators are presented. It is obvious from the figure that the one in the middle is giving the best solutions. It might sound strange that in this case the starting solution is always the same. In this initial creator do the moves made in during the first iterations control very much where the algorithm iterates. The solution structure of the final solution from this initial solution creator does vary as much as for the other initial solution creators. So it is not the case that the TS strategy is always iterating to same good solution neighbourhood. This initial creator was used in the rest of the tests. The computation time and the number of iterations needed was most for initial solution creator number 3, then number 2 and finally number 1.

This significant variability in the best solution found with the different initial solution creators, indicates that the TS algorithm implemented here is sensitive to initial solution. This is in some sense understandable with a trajectory algorithm like TS that the quality of the final solution is dependent on the initial solution. The solution improves very fast during its first iterations, so after few iterations is a "good" solution found. This is not the case with for example SA, see section 4.3.3, which approves almost all solutions during its first iterations (before it is cooled down) and move often very far from the initial solution.

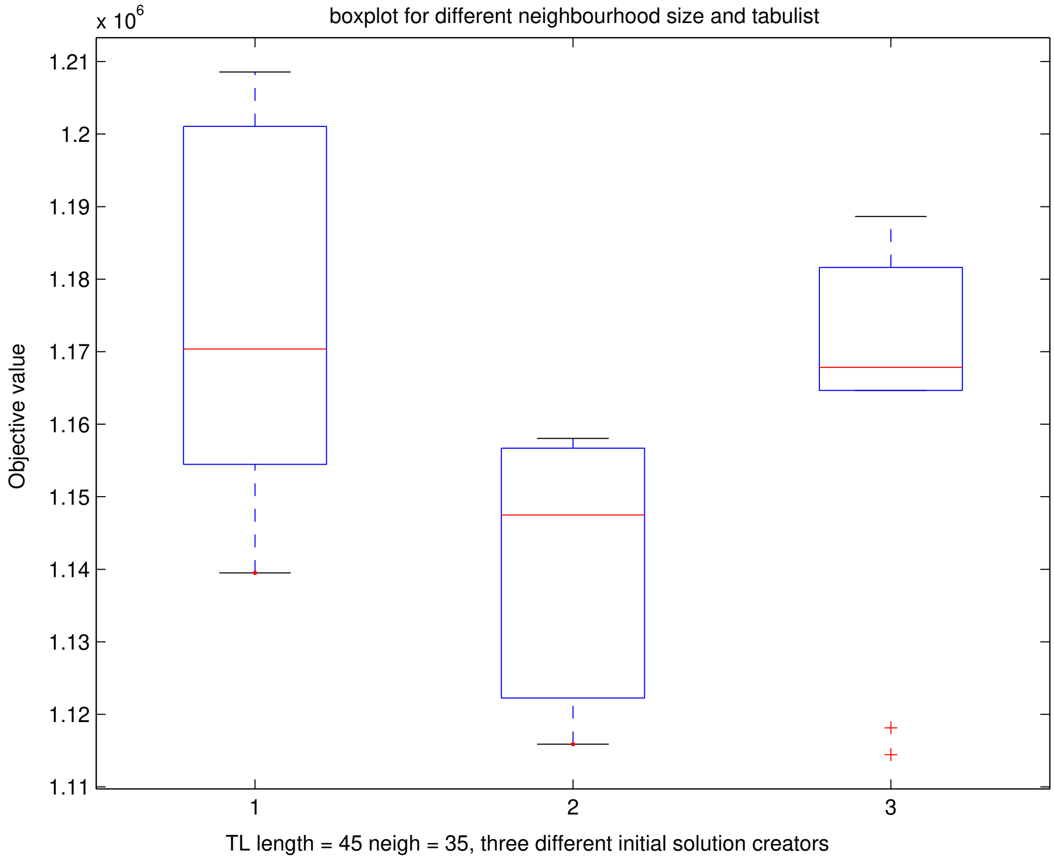

Figure 6.4: Box plot of objective value for different initial solution creators. Leftmost is the first is the one no 1 then so forth.

6.1.3 Different types of tabu lists

The tabu list that was used consists of vectors of length N (number of units) were each element of the vector is the number of times that the unit has been ON. In this section will that TL be referred to as single TL. It was also tested to have another tabu list referred to as time tabu list. That tabu list consist of vectors of length T (number of time periods) were each element of the vector was the number of units that were ON in each time period. This tabu list was only used when a solution was declared "tabu" by the other tabu list. A solution was only rejected if it was declared tabu by both tabu lists. Together are they called double TL. The idea was to decrease the possibility that a solution was rejected that not actually had been visited.

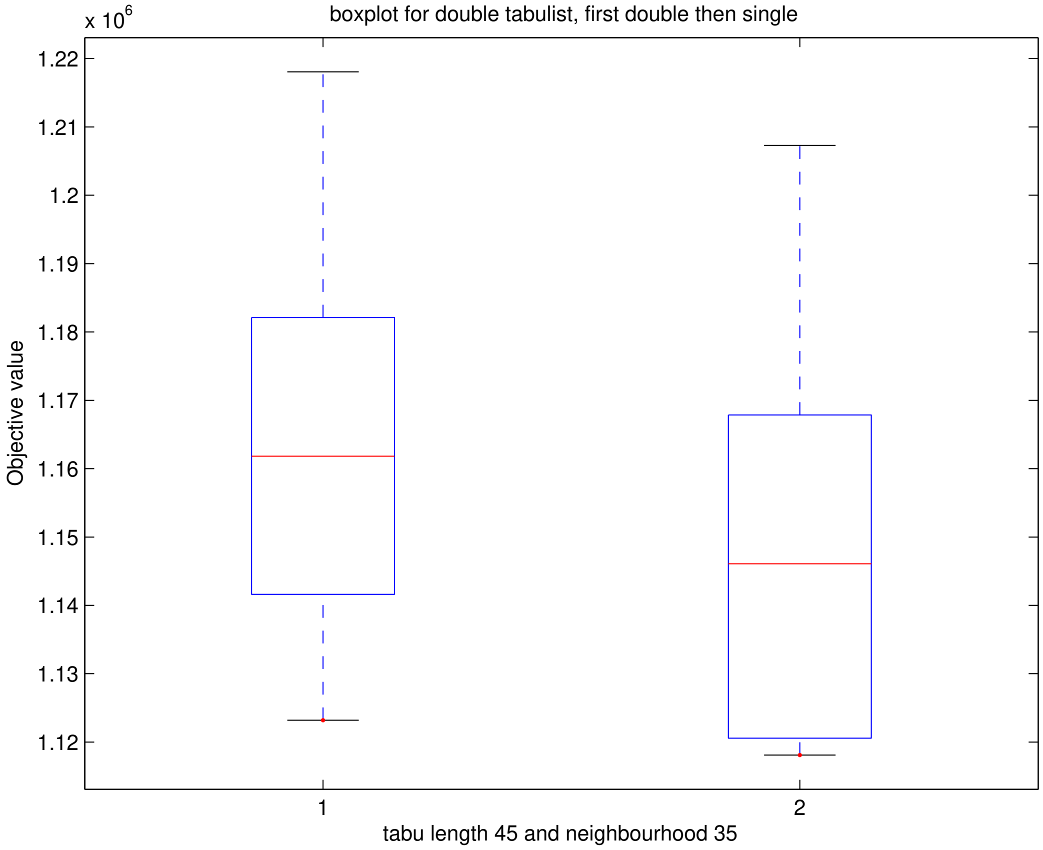

Figure 6.5: Box plot of computational time for different tabu list types. The double tabu list to left and the single one to right.

The results can be seen in box plot on figure 6.5. The one to the right is better and that is the one with single tabu list. The reason can be that the double TL is simply too strict and is not stepping out of local minima enough. The single TL is not only rejecting visited solutions but also solutions in the neighbourhood of visited solutions and that helps escaping local minima.

6.1.4 Stopping criteria

In the TS strategy there were two stopping criteria used simultaneously, see section 5.2.1. The first one is if the number of iterations taken has reached some defined maximum called n_{max}. The second one is if the number of iterations taken since the best objective value was updated has reached some maximum number called j_{max}. Usually n_{max} was set to 40000 and j_{max} to 5000. In over 90% of the runs did the TS strategy stop because j_{max} was reached and that is an indicator of that the TS strategy has started to diverge. To be absolutely sure that this stopping criteria was not too low it was tested to triple it to 15000. The results are in table 6.3. As can be seen in the table is there no significant difference in the best value for the two cases and the best value is in both cases found after similar number of iterations. This indicates that after 5000 iterations without improving best objective value found the algorithm just continuous to diverge. On the other hand the iteration time was much longer when $j_{max} = 15000$.

Table 6.3: Results from test of two different size of stopping criteria.

j_{max}	Objective value	Opt. after # itr.
5000	$1.1486 * 10^6 \pm 30000$	6000 ± 3500
15000	$1.1444 * 10^6 \pm 23000$	5500 ± 3500

6.1.5 Neighbourhood search

The neighbourhood search was described in section 5.1. Before the one described in section 5.1 was used other more simple neighbourhood search strategies were used. They all did work on the same principle, that is change unit status of one unit over some time periods and always preserve feasibility of the solution. Their difference lies in which changes they allow.

The 3 different neighbourhood strategies are:

1. Changes unit status only from the start of a continuous string of same unit status. This is actually the same neighbourhood search as used in [19].
2. Changes unit status from start and also in the middle in a continuous string of same unit status but always just for $Tmin_{down}$ ($Tmin_{up}$) hours. This one is almost as the neighbourhood search used except only allows change from the start and the change is always $Tmin_{down}$ ($Tmin_{up}$) periods.
3. Changes unit status from both ends and in middle and for various number of time period when time periods when changing from the ends. This is the same one as described in section 5.1.

The strategies numbered 1 and 2 here above can be defined as subsets of strategy 3 which is the same one as was described in subsection 5.1. Strategy 3 allows all changes that strategy 2 allows and strategy 2 allows all changes that strategy 1 allows. The results are shown in box plot 6.6. From the figure it is obvious that neighbourhood strategy 3 is best. This shows that though it is a good approach to only change from one end of the continuous string of same unit status it is better to allow change in the middle of it and the end also. This neighbourhood search strategy was then used in all

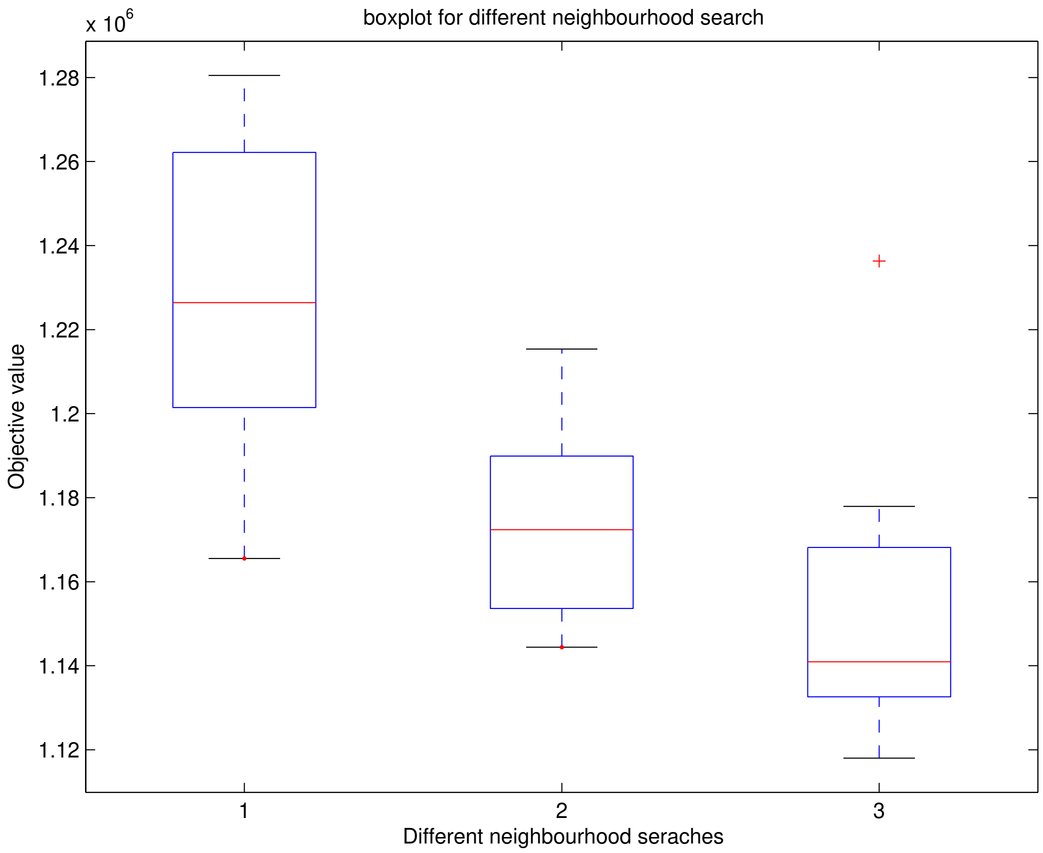

Figure 6.6: Box plot of objective value for 3 different neighbourhood strategies. First is the one that only changes status from the start of a continuous string, the second one changes from start and also in the middle but always just for $Tmin_{down}$ ($Tmin_{up}$) hours, the last one changes from both ends and in middle and for various number of time period when time periods when changing from the ends.

other runs.

6.2 General Concepts

The TS strategy has the characteristic that it finds very fast a rather good solution then tries to improve that solution with help from the TL. For the first iterations does it works almost as a steepest descent method were in almost all sub-neighbourhood $N(x, TL)$ a better solution is found than the current one. Figure 6.7 shows the iteration trajectory of a typical run. As can be seen on the figure the objective value does decrease extremely fast for the first 1000 iterations[2]. Then do almost all sub-neighbourhood search $N(x, TL)$ end with finding a better solution than the current one and few good solutions are forbidden by the TL. Without the TL the strategy would converge after around 1000

[2]the iteration counter counts the number of solution checked but not number of neighbourhood checked.

iteration in objective value around $1.15 * 10^6$, for this case.

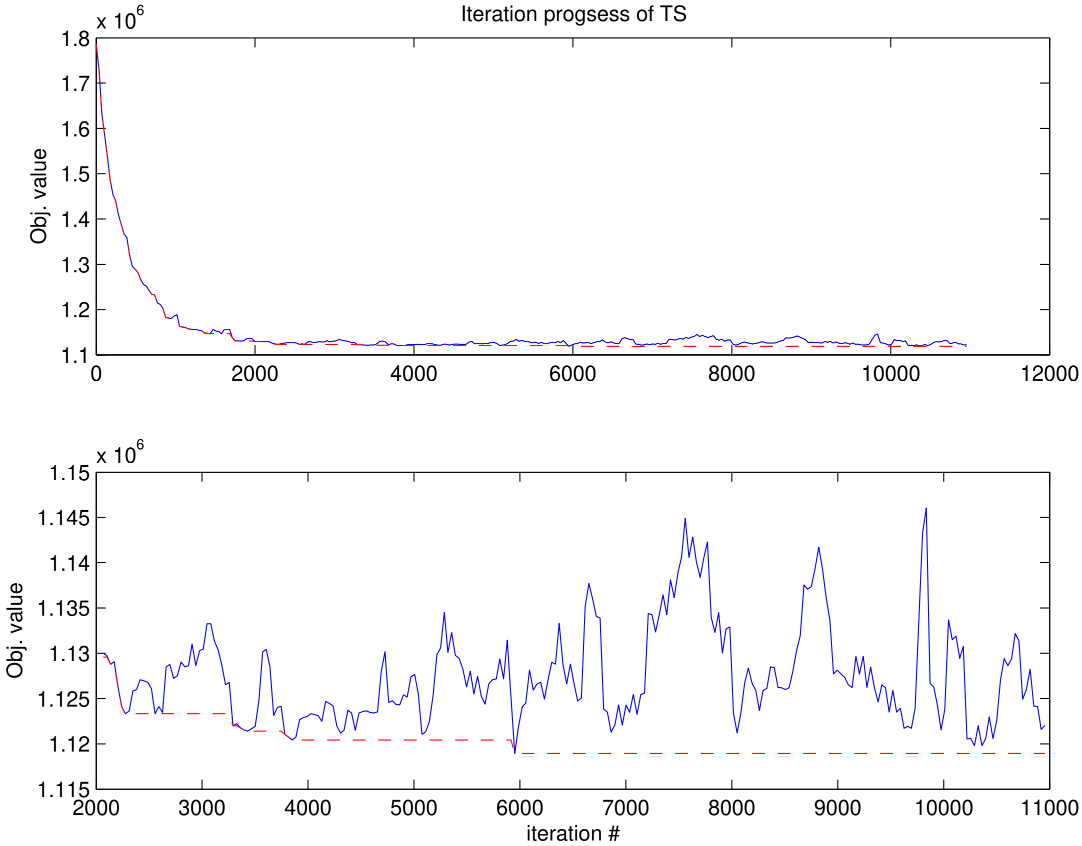

Figure 6.7: Iteration process for a typical TS run. In the lower figure the iteration process is expanded starting at iteration 2000. The solid line is the iteration process and the dashed line is the best found solution.

The lower plot on figure 6.7 shows the trajectory from iteration 2000. This can be called the "active period" of the TS strategy, because during that period the strategy is frequently approving worse solutions and simultaneously stepping out of local minima. During that period many neighbourhood searches do result in that a worse solution being approved than the current one. Of the total iteration process 45 % of the neighbourhood searches do results in that a worse solution is approved and that is mainly because so many worse solutions are approved in the final iterations. It is hard to say whether the strategy is always going to the same low area or finding new low areas. One drawback of TS is that it often does not succeed in locating the best solution in a given low area because it moves to a new solution as soon as the sub-neighbourhood $N(x_n, TL)$ has been searched. At that time it can be that a better solution existed in $N(x_n)$ that was not found.

6.3 Compare with optimal

As has been mentioned before it is possible to solve the UCP with exact methods like B&B at least for small problem instances. As the B&B algorithm is available through GAMS it was straightforward to try that. The implementation is given in appendix ??. In this section the results from GAMS will be compared with the results from TS strategy implemented.

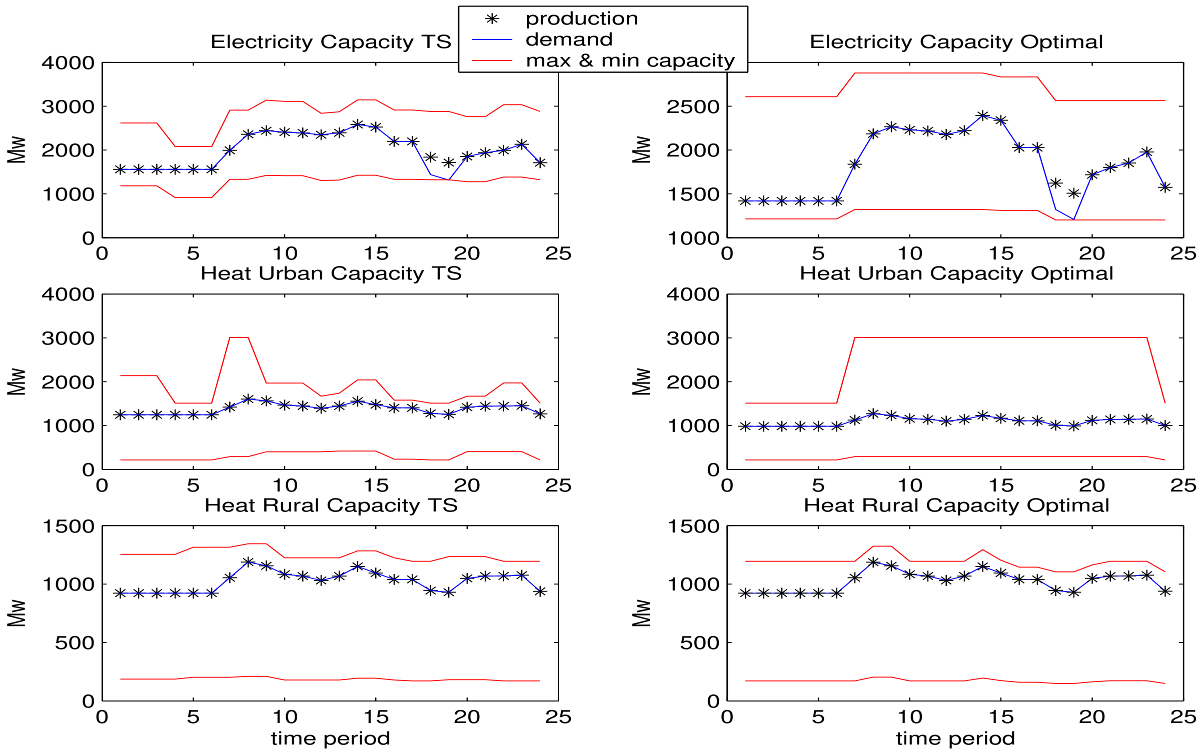

Figure 6.8: Production schedule found with TS to the left and optimal production schedule to right. Demand is plotted with total production capacity of the active units. First electricity, then heat urban finally heat rural

Figure 6.8 shows production schedules from TS and GAMS. At the top is the electricity production schedule, then heat urban and finally heat rural. There for example is too much capacity for electricity and heat urban for the first time periods of the TS solution and an extra unit is turned ON around time period 15 which can be noted in electricity capacity. The optimal production schedule fluctuates less and often is closer to the demand. In the heat urban area the capacity from time period 7 is way too much, that is because a large heat only boiler is turned on, but their start-up cost and minimal capacity is low . Here spinning reserve (SR) is 10 %, because of the SR the maximal capacity is always at least 10 % over demand.

In figure 6.9 the unit status is shown for all units. On it the black colour do mean that active unit and white colour inactive unit. This is the same unit schedule as is shown

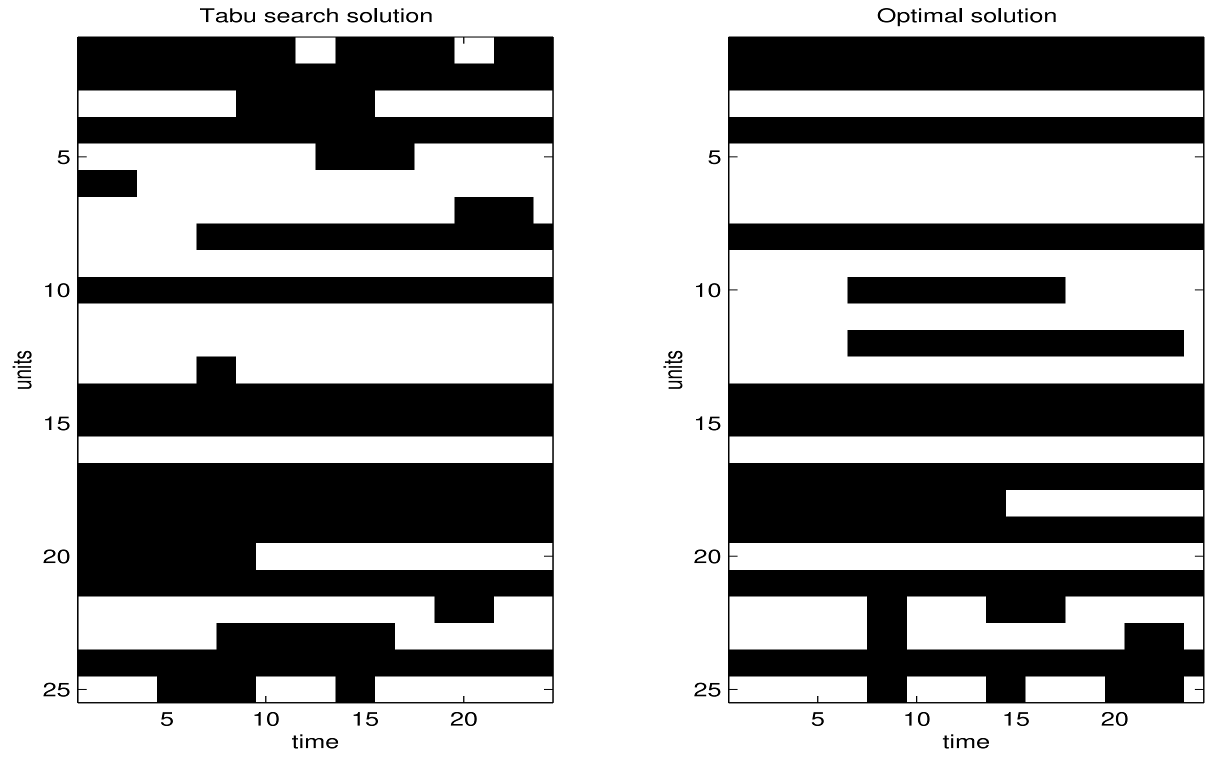

Figure 6.9: The unit status of all units. Black means that unit is active and white that it is inactive.

in figure 6.8. In the TS solution units are more frequently turned ON and OFF. It is though obvious that many of the units that are ON at all hours in the optimal solution are also always ON in the TS solution. This indicates that TS is succeeding in locating some good units. The objective value that TS found here was around 4 % from the optimal, which is around average performance of the TS algorithm.

6.4 Sensitivity to the data

Most of the data originates from the *Balmorel* model. It is solved as a linear programming problem, where the binary variables have been relaxed therefore there are some data that the UCP requires that is not in the *Balmorel*. That is the minimal production capacity, start-up cost and minimal up and down times. A qualified guess was used to compensate for that data loss. The guess was based on information from experience people in the power production field and written literature about power production in CHP plants and power production in general.

Of course are these information not accurate. It therefore was interesting to see how sensitive the problem and the TS algorithm is to change in this data. It was tested to by changing start-up cost, minimal up and down times and minimal production capacity.

The results from these test will be represented in next sections. In the tests in this section the time horizon is 24 hours and each test is composed of 10 runs of the TS algorithm.

6.4.1 Start-up cost

The start-up cost is one of the factors that had to be estimated. The start-up cost was set equal to the cost of producing at unit maximum capacity for 1 time period except for heat only boilers and 2 condensing units with the highest production cost. The reason for this choose is because in heat only boilers it is almost possible to produce heat continuously from zero and it is inexpensive to turn them ON. The start-up cost was also set low in 2 condensing units, but they are designed such that they can be started up fast and inexpensively but it is expensive to produce in them.

In figure 6.10 the results from the test with different start-up cost are shown. In the leftmost box there is no start-up cost, then start-up cost is halved, then normal start-up cost, then doubled stat-up cost and finally four times normal start-up. The little stars under the boxes represent the optimal found with GAMS. The median of the box plot is from 4% to 8 % from the optimal value which is acceptable for a heuristic method. It also seems to be a tendency such that the variance of the solutions gets larger as the start-up cost get larger. When solving the problem with normal start-up cost are there a few units that are ON at all hours while some are only turned on over the peak hours. When the start-up cost is increased the number of time periods that each unit is open increases and the deviations in the solution structure are therefore magnified. This results in more variance of the solutions obtained.

On figure 6.11 the unit status is shown for two different start-up costs. On the left plot start-up costs is halved and on the right one they are four times normal. It can be easily noted in the case where the start-up cost is less, units are more frequently turned on and off. These two solutions gave actually similar objective value.

The start-up cost was implemented such that no start-up cost was paid in the first time period. The reason for this is because it is assumed that the unit status in the first time period is the same as in the time period before (that is the time period before the optimization starts). Cause of this it often is desirable to have a unit active from first time period even though its capacity is not needed until later. This especially is the case when the stat-up cost is increased.

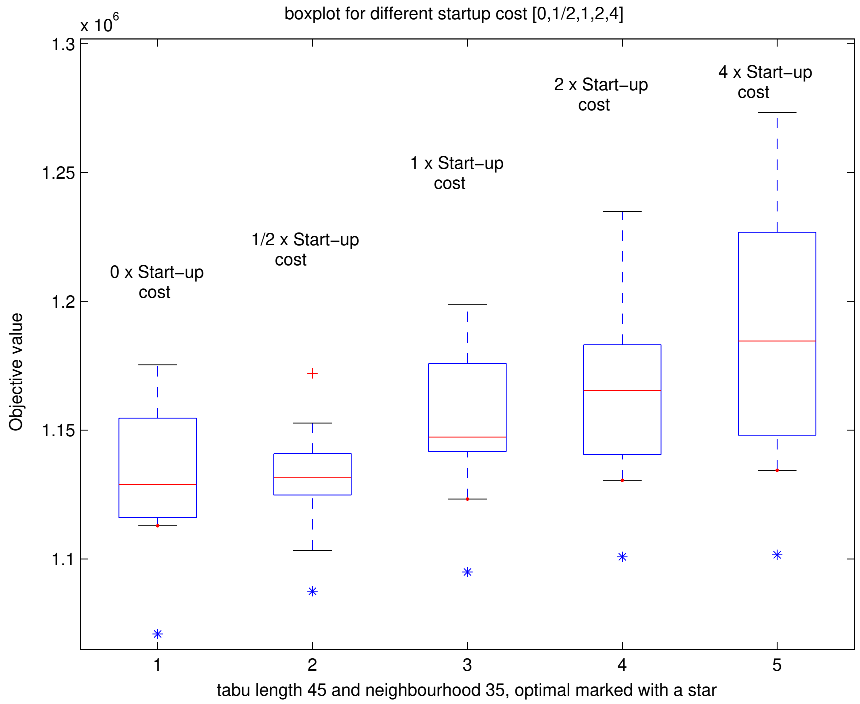

Figure 6.10: Box plot for different start-up cost. Most to left there is no start-up cost, then start-up cost is halved, then normal start-up cost, then doubled stat-up cost and finally four times normal start-up. The optimal value found with GAMS is marked with a asterisk.

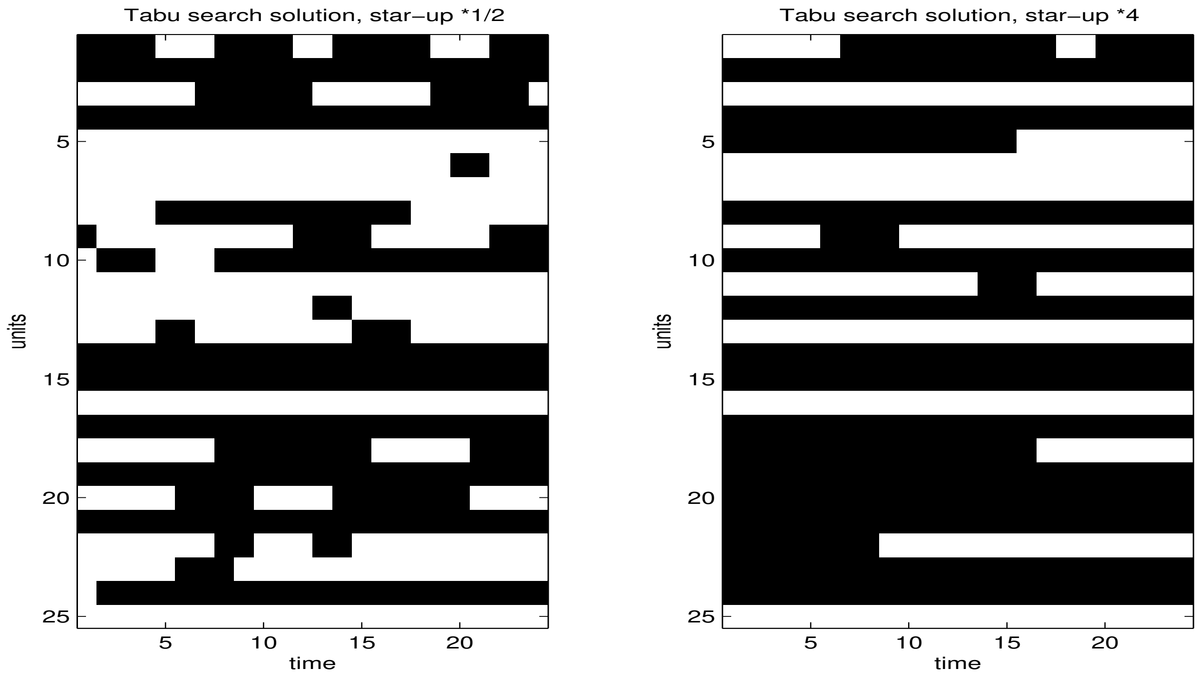

Figure 6.11: Unit status of each unit. Black means that unit is active and white that it is inactive. On the left figure the stat-up cost is divided by 2 and on the right one it is multiplied by 4.

6.4.2 Minimal up and down times

Minimal up and down times also had to be estimated. Similar principles were used when evaluating them as for the start-up cost. Units that in principle can be started up rapidly have short minimal times. While unit that have large maximal capacity did have longer minimal up and down times. The minimal up and down times can be seen in table 6.4. It should be noted that minimal up and down times are here always equal.

Table 6.4: Minimal up and down times for different units with different maximal capacities. First column shows the unit type, second column the maximal capacity of unit and last column the corresponding minimal up and down times.

Unit type	Maximum el. capacity	minimal [h]
Extraction	All	2
Back pressure	>400 [MW]	5
Back pressure	<400 [MW]	4
Heat only	All	2
Condensing	>600 [MW]	5
Condensing	<600 & >500 [MW]	4
Condensing	<500 [MW]	3

To test the sensitivity for different minimal up and down times it was tested to change them. The tested cases can be seen in table 6.5. In test 1 are the minimal up and down times are neglected by setting them all to 1. The results can be seen on figure 6.12. As expected the objective value did slightly decreases with decreasing minimal up and down times, but the change was minor (less than 1 %). The exception is the case where all minimal times are equal to 1. In that case TS finds worse objective values than for longer minimal times. The reason for this is because the neighbourhood search strategy searches only in the feasible region with respect to minimal up and down times. When minimal up and down times are neglected then $N(x_n)$ is much larger and it is harder to located good solutions. In that case some other neighbourhood search strategy would be more appropriate, which would allow smaller changes in each step.

Table 6.5: The different minimum up and down times that were tested. In test 3 are the minimal up and down time unchanged.

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

On figure 6.13 the unit status is shown, minimal up and down times are all equal to 1 on the left figure and on the right figure they are normal plus 2 hours. Both solutions

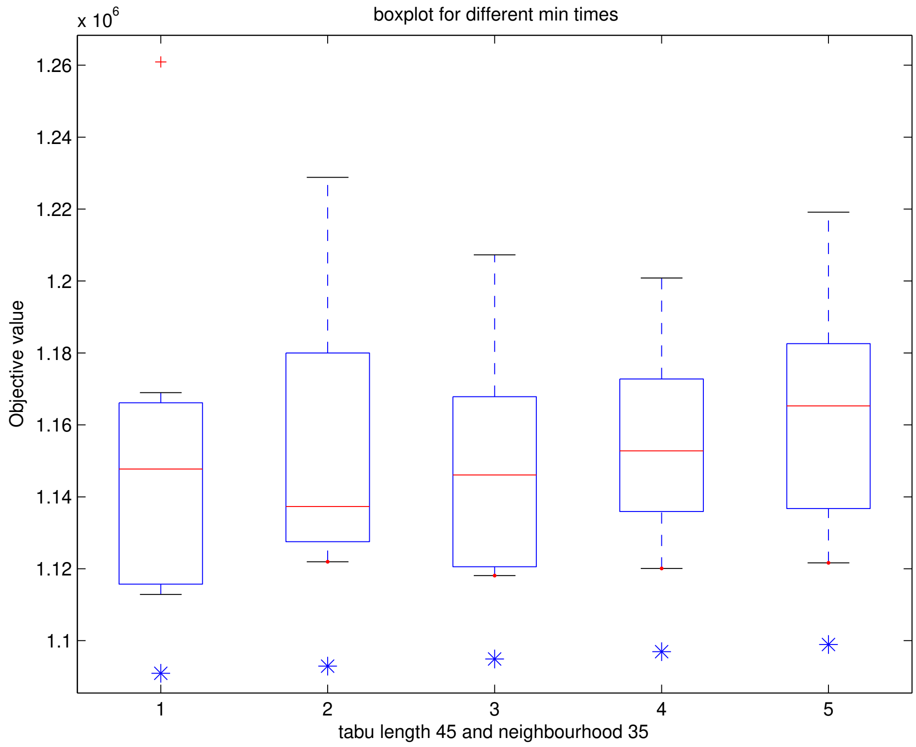

Figure 6.12: Box plot of different minimal up and down times. 1: All minimal times equal to 1 2: They are normal minus 1 3:Unchanged 4: Normal plus 1 hour 5:Normal plus 2 hours. Optimal value found with GAMS is marked with a asterisk.

are obtained with the TS strategy. The difference can be noted in figure 6.13 where few units are only turned on for only one hour in the plot to the left while all units are on for a longer time on the plot to the right.

The minimal up and down times are implemented such that they do not apply when unit is turned ON in the first time period. The reason for this is because it is assumed that the unit status in the first time period is the same as in the time period before (as was done for start-up cost). They do also not apply in the end of the time horizon as it is assumed that these units will be kept open in the coming time periods (after the optimization period is over). Because of this can it be noted in figures 6.11 and 6.13 that these constraints are often violated in the beginning and the end of the time horizon.

The number of iterations needed did not change significantly with change in minimal up and down times, but there was a remarkable tendency to reject more solutions as the minimal up and down times were decreased. This can be seen in figure 6.14. When the minimal times are smaller the change in each step usually is small compared to when the minimal times are larger. When the change in each step is smaller it is more likely that a solution is rejected that has not actually been visited. Therefore more solutions are rejected when minimal up and down times are shorter.

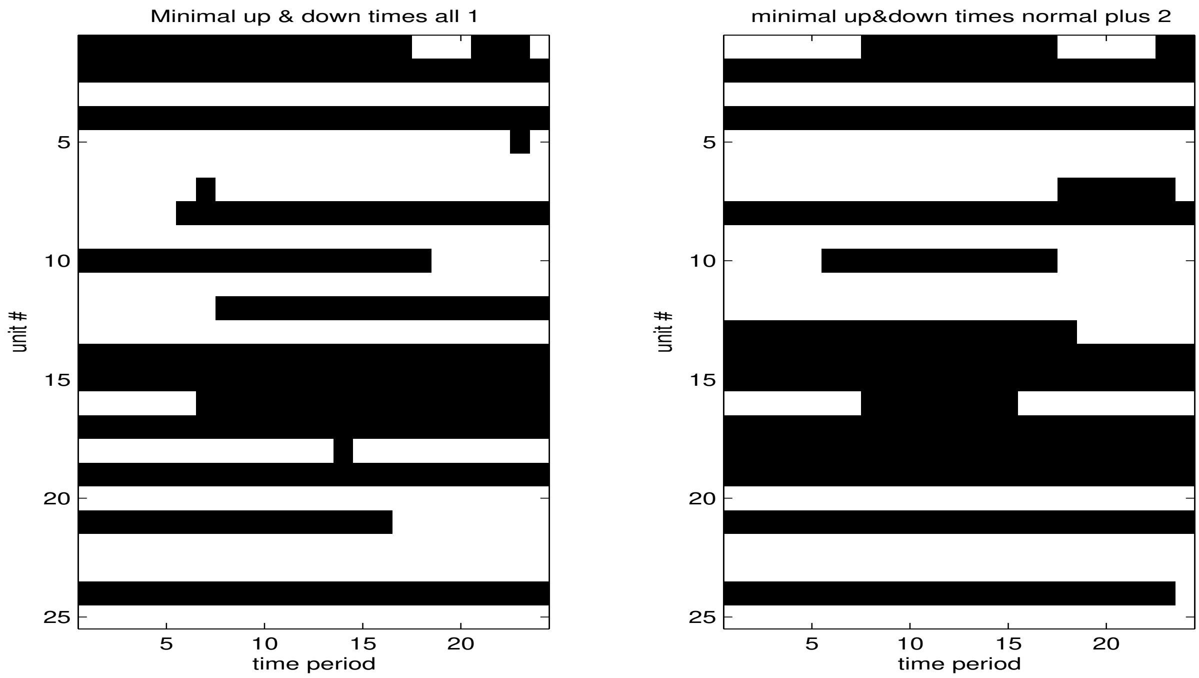

Figure 6.13: Unit status of each unit. Black means that unit is active and white that it is inactive. Here are minimal up and down times on the left figure all 1 and on the right picture normal plus 2 hours.

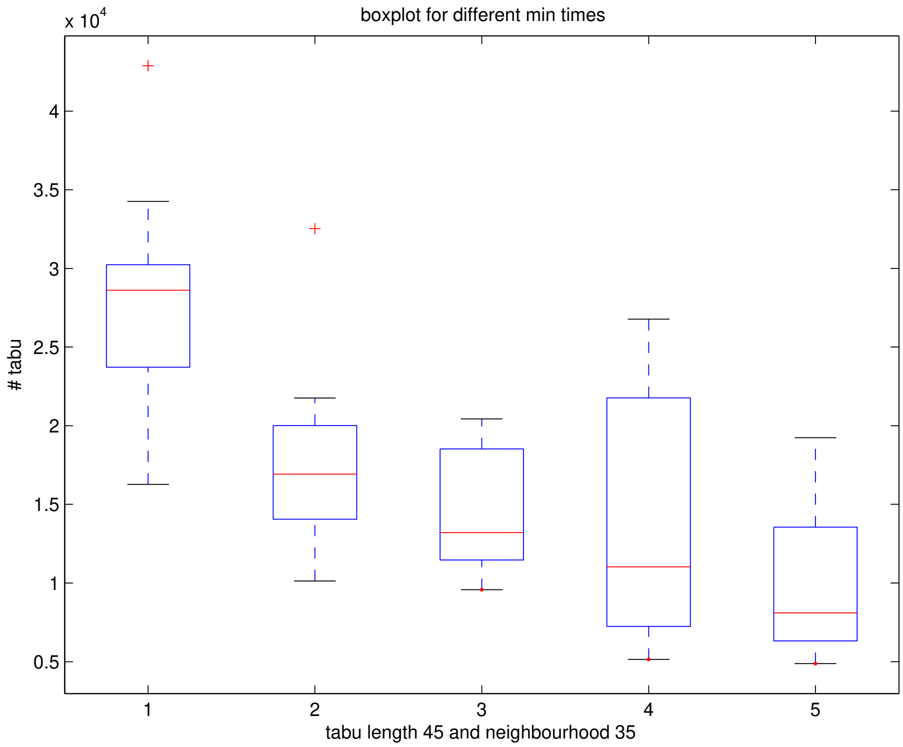

Figure 6.14: Number of solutions rejected by the tabu list. 1: All minimal times equal to 1 2: They are normal minus 1 3:Unchanged 4: Normal plus 1 hour 5:Normal plus 2 hours.

6.4.3 Minimal Capacities

The minimal capacities for each unit can be seen in table 6.6. It was tested to change the minimal capacity in each unit. The tested change was to increase and decrease production capacities by 50 %. Not much change could be noted in the objective value, similar average and best values were obtained. Not same change was noted in unit status as when changing start-up and minimal up and down times.

Table 6.6: Minimal capacities for different maximum capacity of units. Heat only boilers are reported specially, but as they in practice can produce almost continuously from zero the minimal capacity were set low in them. The lower row represents the percent of the maximal capacity that the minimal capacity was set to.

Maximum Capacity	> 500 [MW]	> 200 [MW]	< 200 [MW]	Heat only boilers
Min % of max	50 %	40 %	25 %	10%

6.5 Solving larger problem

The problem that has been solved in this chapter is a UCP with 25 units and a time horizon of 24 hours. That problem includes 24*25 binary variables or 600 in total. In practice it is not practical to solve the UCP for more than at most 2 or 3 days. The reason for that is because there are just too much variability over longer time periods so it is possible to solve the UCP. For example can the demand not be predicted with much accuracy for longer time periods and the units also are subject to unexpected malfunctions

The TS strategy can solve larger problems than for one day. It was tested to solve the same problem for 2 and 4 days, and compare the results with the optimal. Some of the production units in the system have extremely large capacity. Many of them are not set together of one large turbine (and boiler) but of a few small ones. It was therefore also tested to spilt some of the larger production units in 2 or 3 smaller units, with smaller capacity each. This was done in such a way that the total capacity did not change but the number of units increased from 25 to 34. This problem is also solved for 1, 2 and 4 days.

6.5.1 Solving over few days

What demand to use is one of the major factors when solving the problem over more than one day[3]. One possibility would be just to use the same demand as for the first day. That is not practical as demand not only fluctuates within a day but also between

[3]demand was discussed in general in chapter 2.

days (and between parts of the year). It also makes the problem easier to solve when same demand is used over and over. Therefore it was decided to have some fluctuation between in the demand between days. In practice there is more fluctuation in the electricity demand between days than heat demand. But more fluctuation in heat demand between different parts of the year. The fluctuation was set to ± 10 % for electricity and ± 5 % for heat.

Table 6.7: Optimal value found with TS and optimal value, for different sizes of problems. The best TL length and number of search solutions in each neighbourhood is also reported.

	Average	Std. dev.	Optimal*	% difference	TL length	$\lvert N'(x)\rvert$
1. day	$1.15 * 10^6$	30000	$1.09 * 10^6$	5.5 %	45	35
2. days	$2.77 * 10^6$	20000	$2.74 * 10^6$	1.1 %	45	35
4. days	$4.89 * 10^6$	170000	$4.48 * 10^6$	9.0 %	50	35

*found with GAMS

The results from the test are given in table 6.7. In the table the mean value is found with the TS strategy reported with its standard deviation, the results are also compared with the optimal value found with GAMS. The results in general are quite satisfactory only around 5.5 % from optimal when solving for one day. Solutions for two days are not further from the optimal than the solution for one day. A possible explanation of this is that it helps the problem is periodic. Units that are "good" on the first day will also be "good" on the second day. The neighbourhood search strategy allow changes over many time periods of a unit in each iteration and that helps locating these "good" units. The solutions that are obtained when solving for 2 days are actually best, only 1.1 % for the optimal. A possible explanation is that the random walk of the TS during its "active period", see section 6.2, it is easier to locate a good solution for this size of problem.

The solutions obtained for 4 days are on the other hand considerably worse than for the smaller problems. The solution is as high as 9.0 % from the optimal in that case. In practice do heuristic method often return solutions that is often up to 10 % from optimal. In that sense this is not too surprising. When the problem size is larger the possibility of rejecting a solution by the TL, decreases. Therefore larger problems often require longer TL to achieve good solutions. That was the case for 4 days where the length of the TL that gave best solutions was 50. For 2 days the best TL length was the same as for one day or 45.

When the problem gets larger the overhead involved in copying matrices gets larger. There is also an overhead involved in comparing to the TL. The UC solution matrix is twice as large for 2 days than for 1 day. That does Of course increase the time overhead.

This overhead can be noted very well in the execution time of the strategy. In figure 6.15 the execution time is plotted versus size of the problem. The iteration time seems to increases linearly with size of the problem. That is understandable as the size of the UC solution matrix increases linearly with number of days.

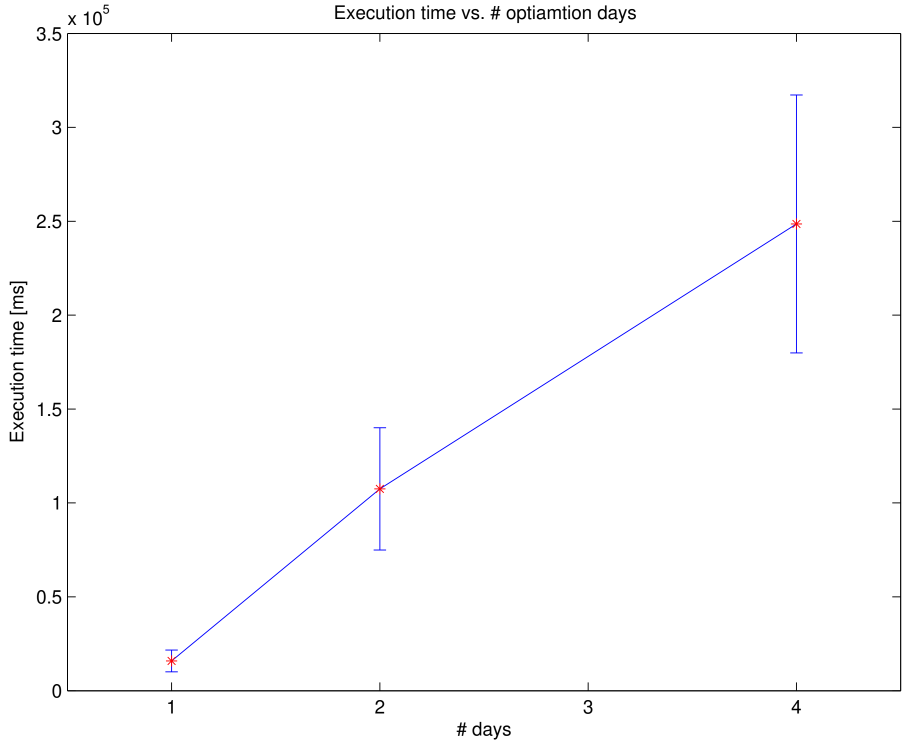

Figure 6.15: The execution time plotted versus different size of the problem. The bars at each objective value, show the spread in the execution times.

The execution time also increased drastically in GAMS as the problem size became larger. That can be seen in table 6.8. The execution times are not too extreme for 1 and 2 days but for 4 days the execution time explodes, where the execution time was around 960 s = 16 minutes. The TS strategy proved to be much better where the execution time are never more than 8 minutes.

Table 6.8: Execution time for GAMS.

Days #	1	2	4
Time [s]	24	40	960

6.5.2 Splitting units

As mentioned before most of the large units are made of a few boilers (and turbines) that can be turned ON individually. To simulate this some of the larger units were splitted into 2 or 3 separate units. The rule if thumb that was used was to split units until all units had capacity under 300 [MW]. To increase variation between units the total capacity was not divided equally between the units. By this the number of units increased from 25 to 34. The data can be seen in appendix ??. The purpose of increasing the number of units and to see if the TS strategy is capable of solving systems where number of units have been increased. Just as it is for systems where the time horizon has been expanded. It was also expected that GAMS would have problems finding an optimal solution to this problem.

For each unit that was divided into smaller units, a random variation was added to the production cost and the working area parameters to make the units distinguishable. The tests were done in a similar way as the tests in previous section.

Table 6.9: Best objective value found with TS and optimal value, for the spitted problem. The best TL length and number of search solutions in each neighbourhood is also reported.

	Average	Std. dev.	Optimal*/lower bound	% difference	TL length	$\lvert N'(x)\rvert$
1 day	$1.06 * 10^6$	18000	$1.03 * 10^6$	2.9 %	45	35
2 days	$2.60 * 10^6$	30000	$2.56 * 10^6$	1.6 %	50	35
4 days	$4.58 * 10^6$	50000	$4.24 * 10^6$	8.0 %	55	30

*found with GAMS

The results from the tests can be seen in table 6.9. These are rather similar results as they, which were obtained for the un-spitted problem. Here the optimal is considerable closer to the optimal for 1 day. It seams to help the algorithm when the unit are more and the capacity of each is smaller. For 2 days the result are slightly worse than for the un-splitted problem but for 4 days it was slightly better than before, but the difference is relatively small in both cases. GAMS was able to solve the problem for one and two days to optimality. For 4 days it stopped with a duality bound equal to 0.9 %. In table 6.9 the lower bound is represented when optimal could not be found. The optimal solution can be up to 0.9 % higher than the lower bound. It would be interesting to see what would happen if the problem size would be enlarged further if the TS strategy would give worse solutions or similar as before. Unfortunately there was not time to try that. As in previous section larger problem instances require longer TL, here dose the best TL length increases by 5 for each increase in problem size. Same tendency was noted in increasing of execution time as in last section.

6.6 Comparing TS and SA

A fellow student implemented a SA algorithm to solve the same UCP. Both the TS described here and the SA used the same method to solve the EDP, as was described in chapter 3. SA is also a local search heuristic as was listed in chapter 4. The SA used same neighbourhood search strategy as used here. The detailed description on the SA implementation can be found in [12].

6.6.1 The iteration process

In figure 6.16 a typical iteration process for TS and SA can be seen. The TS strategy iterates very fast to a good solution. As for the first iterations is not hard to find a good solution in the neighbourhood of the current one and the TL forbids few good solutions. After around 1000 iterations usually a rather good solution is found with objective value around 10 % from optimal. After that it is tried to improve the best found solutions with help of the TL to step out of local minima. The total iteration process almost forms a large L. Best objective is usually found after around 5000 iterations. Here are 150.000 iterations taken because SA usually needs so many iterations to converge. It is interesting to see that the best objective value that TS finds does not improve much by taking so many iterations, therefore could it be that TS is again and again going to the same solution area.

The iteration process for the SA is quite different. SA always selects only 1 solution in each neighbourhood at random, as can be seen in section 4.3.3. In first iterations SA approves almost all solutions. During that period it iterates to a value around $1.4 * 10^6$, which is around average value of all solutions for the problem of this size. For the next 3000 iterations the objective value is around $1.3 * 10^6$ to $1.4 * 10^6$. After that the process has cooled enough down so fewer worse solution are approved. For the last 80000 iterations there is little change in the objective value. The final value that SA finds is considerably lower than the one TS finds.

It is finally worth mentioning that the execution time for TS is much longer than for SA for same number of iteration. SA is around 4 time faster. That can be explained by the overhead involved in copying solutions matrices and comparing solutions with the ones in the TL.

6.6.2 Sensitivity

Same tendency was noted in both algorithms when the minimal up and down times were changed. The objective value decreased slowly with decreasing minimal up and down times. Both algorithms performed worse when there wore no minimal up and down

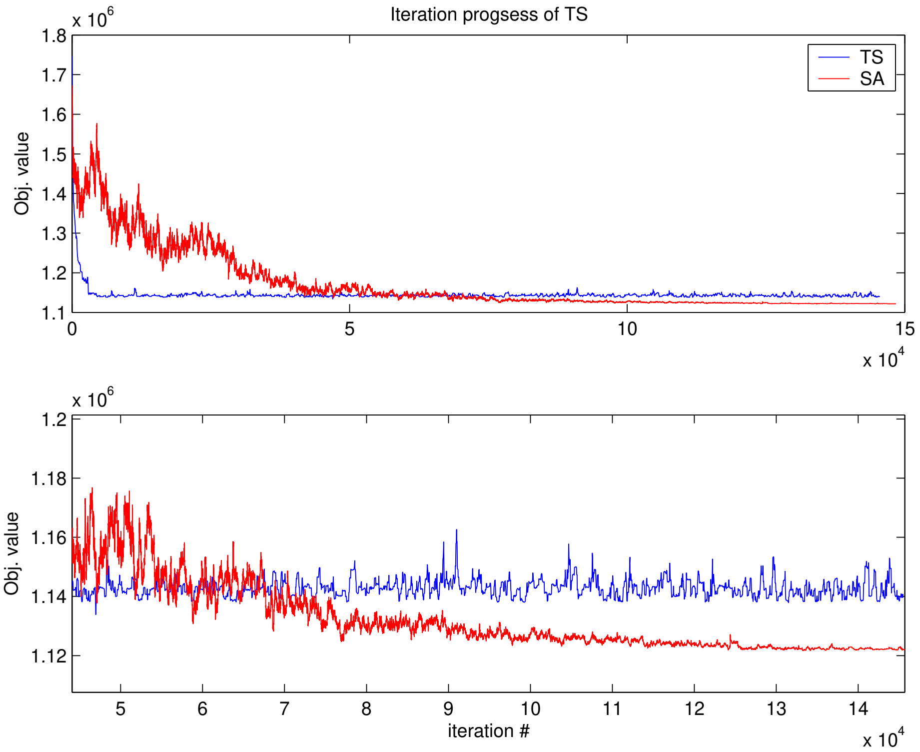

Figure 6.16: Iteration process for TS and SA. On the higher plot is the total trajectory shown and on the lower is the trajectory from iteration 50000 is shown.

times (same as set all to 1). That is not surprising as both used same neighbourhood search strategy.

Start-up cost had similar effect in both cases. Objective value increased with start-up cost and variance was larger when start-up cost was higher. A difference was noted in sensitivity to initial solution, TS were more sensitive to initial solution than SA.

6.6.3 Difference in Objective Value

In figure 6.17 the objective values found with TS and SA are compared. The bars at each value show the range from best to worst solution. The objective value is divided by the number of days to make it easier to compare visually. As can be seen in the figure there is not much difference in the objective value found with the two algorithms found for two days. But for one and four days there is greater difference. SA finds both better objective value at average and the range is less. The TS does however also sometimes find as good solutions as the best ones that SA finds, but it happens more seldom. For example does SA finds solution which are around the best ones in every 10 run while TS finds them in around every 20 -30 run. So it is not the case that TS not succeeds finding the best solutions. The range can in some sense be explained by the sensitivity to initial solution in TS. The different in best objective value found by the two algorithms is possibly because of TS difficulty to locate the best value within a

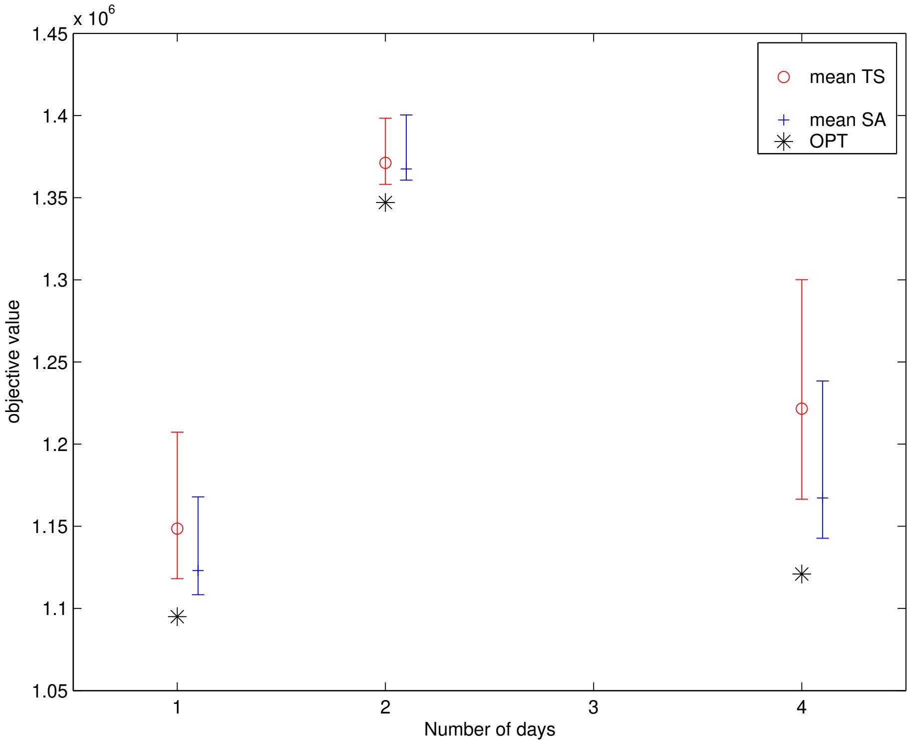

Figure 6.17: Comparing of objective value found with TS and SA. The error bars around each value show the range of the data. It should be noted that the objective value is divided with the number of days in the problem so it is easier to not difference in quality of the solutions. The optimal value is marked with a big star.

good solution area. SA is "cooled" slowly down and approves almost all solution in the beginning of the iteration progress and is able to locate better solution area. In the end of the iteration process it almost only approves better solutions so it can (more easily than TS) find the best solution in a good solution area.

6.7 Conclusion

In this chapter the unit commitment problem (as formulated in chapter 2) has been solved using Tabu Search strategy. The time horizon has been from 24 - 96 hours and the number of units 25 and 34. The results are compared with an optimal or lower bound solution obtained with Gams. The results obtained were quite satisfactory, when solving smaller problem instances that are for 1 and 2 days. In those cases the average performance was 1% - 6% from the optimal, depending on the problem at hand. Solutions obtained were up to 10 % from optimal for 1 day, but up to 5 % for two days. The average was closer to the lower end of the range in all cases.

The case where the problem size is 4 days is quite different. In that case the solutions obtained are considerably worse or at average 9.0% and 8.0% from optimal for original and splitted problem, respectively. Best solution were around 2 % from optimal/lower bound in both cases. It is though not completely dependent on the size of the problem as the algorithm seams to solve the splitted problem better than the un-splitted one and netter for two days than one. It would be interesting to see what would happen when solving larger problems.

The neighbourhood search gave quite satisfactory results. It allows rather great change on the solution in each iteration. It worked considerably better than the original which is number 1 in the test in section 6.1.5.

The algorithm showed tendency to be sensitive to initial solution, which increased the variability in the solutions obtained. The best results were obtained when the starting point was always the same or with all units on at all hours. This does not skew the results as all solutions can easily be obtained from this solution. To use a metaphor this is like starting on the top of a mountain and then the first steps control much about in which valley the tour ends.

The experiment with the data that is *start-up cost, minimal up and down times and minimal capacity* showed that a increase in the start-up cost was most effective. A change in the minimal capacity did not change much. A change in the minimal up and down times did not affect the objective value much but did affect the unit status. The algorithm found worse solutions when there were no minimal up and down times, but the reason for that lies in the way that the neighbourhood search strategy searches. The start-up cost had considerable affect on the unit status and the objective value. It was interesting that there was a tendency to get larger variance in the objective value when the start-up cost was increased. In general it can be concluded that the performance of TS is not strongly dependent on the data.

The problem was solved with GAMS for all problem sizes. GAMS was not able to solve to optimality when some units had been splitted and the time horizon was 4 days. This indicates that the solver is on the verge of being able to solve larger problem. If larger problems were solved the duality cap would only increase. In those cases it would then be better to use a metaheuristic method such as TS, which are not as affected by the problem size.

The TS algorithm was finally compared with a SA algorithm implemented by a fellow student working on the same project. The results can be seen in section 6.6. Both algorithm showed good results but SA obtained slightly better results at average. There is no reason for that SA should show better results than TS, but that is often the case with metaheuristic methods that one method works better for some problems than other.

It would be possible to build a hybrid method of TS and SA. As was noted in section 6.1.1 did the TS not work properly when the number of solution in the sub-neighbourhood $N'(x_n)$ were one. Here arises a possibility to use the SA approach to only approve solutions that are better that the current one, and approve worse solutions with some probability. But simultaneously use the TL to avoid cycling and local minima.

Chapter 7

Conclusion

In this report the use of a TS algorithm to solve the UCP problem for a CHP system has been reported. The problem was based on the *Balmorel* model but for that model a large amount of data has been collected. In the beginning of the work of this project a considerable effort went into sorting the data from the *Balmorel* model. That model is based on linear programming entirely where the binary variables have been relaxed. For that reason there was considerable amount of data that is necessary to formulate the UCP missing. A qualified estimate was therefore used to compensate for that.

The UCP is a combinatorial optimization problem. When selecting a method to solve the UCP there were two things that were most important concerning the choose. They are:

- The method have to be capable to solve large problem instances.
- The method should work considerably fast and produce near optimal solutions

The two approaches that were given most thought were use of exact methods like branch & bound or using heuristic methods. As B&B is available in GAMS it was therefore straightforward to use that. Exact methods have the positive property that they produce optimal solution, but often the computational time is considerable. Often the optimal solution can not be found in B&B when the problem size gets larger. Heuristic methods usually work relatively fast and the quality of the final solution is usually good. Heuristic can also handle large problem sizes. It was decided to use TS to solve the combinatorial part of the UCP. The reason was that it is relatively easy to implement and the iteration process rather transparent. It also sounded interesting to see the TL work in action.

For every combination found by the UCP the EDP had to be solved, and in that sense it can be referred to as the objective function for the UCP. Solving the EDP is expensive function evaluation compared to many other combinatorial problems e.g. the travelling salesman problem. The EDP here is formulated as large linear problem were a complicating factor is that production of heat and power is related. After it was decided to solve the combinatorial part with heuristic method, the decision that had to made was which method to use to solve the EDP. The first idea was to use exact linear programming. The result from that experiment was that it was simply too slow. Therefore was it decided to use some heuristic to solve the EDP. The execution time of the heuristic method was around 1ms, which is 100 times faster than the linear programming. That was very promising as the EDP has to be solved thousands of times for the UCP.

The heuristic method used to solve the EDP is described in 3. It works on rather simple principle. The cost of produce in each active unit was evaluated. By using the ratio of production cost versus production output. Then the production is increased in the least expensive unit as much as possible. This is repeated until demand is satisfied. This is done for all time periods. The starting point was at the minimum capacity in each unit. The results of this approach were very nice. The most error was under 4% from optimal, but usually the error was much less or 1-2 %.

The neighbourhood search that TS uses is described in chapter 5. It only searches in the feasible solution space and allows rather big change on the solution in each iteration.

It was tested to experiment with the TS algorithm sensitivity to the data. There it was considered most interesting to experiment with the factors that had to be estimated. Of those factors were two things most interesting. That is the increase in the average objective value obtained when there were no minimal up and down times. But that can be explained by the way the neighbourhood strategy searches. Secondly the start-up cost had considerable effect on the solution, both on the objective value and the solution structure. In general it can be concluded that the performance of TS is not strongly dependent on the data.

The TS algorithm was tested for problem of size with time horizon from 24 to 96 hours and with 25 and 34 units. The results were compared with optimal and the results were from 1 % up to 9 % from the optimal depending on the problem size. When solving for 24 and 48 hours the average was always less than 5 % from optimal which is very promising. The execution time of TS did increase considerably as the size of the problem increased. That can be explained by the increase in copying of solutions and comparing with the TL. The results were worst when solving for 96 hours. It is impossible to say what the solutions would be if a larger problem was solved. For example the difference from optimal was less when solving for 2 days than 1, so it can be that larger problems would be solved even better.

Finally the conclusions can be made in few points.

- the EDP heuristic is well suited to determine the quality of different unit combinations in relatively short computational time.

- Experiments with the data indicates that the TS algorithm is not very dependent on the data.
- The method of using TS with the EDP heuristic works well when solving the UCP of different size in reasonable computational time.

As the EDP gives good results and the TS strategy is satisfactory. This combination could be a valid method to solve larger problem instance. Especially as GAMS is on the edge of not being able to solve the larger problems.

Bibliography

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

[2] R. Battiti and G. Tecciolli *The reactive Tabu Search.* ORSA Journal on Computing, 6,2.

[3] C. Beltran and F. J. Heredia *Unit commitment by Augmented Lagrangian Relaxation: Testing Twp Decomposition Approaches.* Journal of Optimization Theory and Applications. Vol. 112, No. 2, pp. 295 - 314, February 2002.

[4] A. Borghetti et. al. *Lagrangian Relaxation and Tabu Search Approaches for the Unit commitment Problem* IEEE Porto Power Conference 10th - 13th September, Porto, Portugal.

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

[6] 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

[7] 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.

[8] Thomas H. Cormen et. al. *Introduction to algorithms, second edition.* The MIT press, 2001.

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

[10] F. Glover *Future paths for Integer Programming and links to Artificial Intelligence.* Computers & Ops.Res. 5:533-549.

[11] F. Glover and M. Laguna *Tabu Search* (1993).

[12] Thorhallur I. Halldorsson. *Unit commitment for combined heat and power production, using Simulated Annealing*, IMM-EKS-2003-X.

[13] Elin Hallgrimsdottir. *Optimizing combined heat and power plants in Copenhagen.* IMM-EKS-1998-19

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

[15] Magnus Hinderberger, René Victor Valqui Vidal, *Tabu Search - a guided tour.* Control and Cybernetics, vol. 29 No. 3, 2000

[16] Walter Hobbs et. al. *An enhanced dynamic programming approach for unit commitment.* IEEE Transactions on Power Systems, Vol. 3, No. 3, August 1998

[17] 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.

[18] Chun-Lung and Nanming Chen *Direct Search Method for Solving Economic Dispatch Problem Considering Transmission Capacity Constraints*, IEEE Transactions on Power Systems, vol. 16. No 4, November 2001

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

[20] A. H. Mantawy, Youssef L. Abdel-Magid, Shokri Z. Selim,*A new genetic-based Tabu Search algorithm for unit commitment problem*, Electric Power Systems Research, vol. 49 p. 71 - 78.

[21] 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.

[22] Hiroyuki Mori, Osamu Matsuzaki, *Embedding the Priority List into Tabu Search for Unit Commitment*, 0-7803-6672-7/01 2001 IEEE.

[23] 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.

[24] Pang. C. K. and Chen H. C. *Optimal short-term thermal unit commitment* IEEE transactions on PAS, 1976, 95(4), 1336-1346.

[25] Pang. C. K., Sheble G. B. and Albuyeh F.*Evaluation of dynamic programming based on methods and multiple area representation for thermal unit commitments* IEEE transactions on PAS, 1981, 100(3), 1212-1218.

[26] M. Pirlot *General local search heuristic in Combinatorial Optimization: a tutorial* Faculté Polytechnique de Mons rue de houdain 9 B-7000 Mons- Belgium

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

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

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

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

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

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

[33] Jorge Valenzuela and Alice E. Smith. *A Seeded Mimetic Algorithm for Large Unit Commitment Problems.* Journal of Heuristics, 8: 172-195, 2002.

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

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

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

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