

Final Examination

MSc SENSE

**Implementation of the Optimization Problem for an
Energy Management System**

MASTER THESIS

Author: Manuel Ostertag
Supervisor: Oriol Gomis Bellmunt
Company: GreenPowerMonitor
Handed in: June 2017



Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona





Review

This master thesis project contributes to the development of an energy management system (EMS), whereby the thesis is part of a bigger project carried out by the Barcelona-based software company GreenPowerMonitor. The developed EMS coordinates the different components of a hybrid power plant by repeatedly solving an optimization problem over one day and computing the optimal setpoints of the components' controllers for the next minutes. In the context of the EMS project, optimality is defined as maximizing the amount of used solar power while minimizing the battery losses for several future generation and consumption scenarios over the course of one day.

This thesis project focuses on examining ways to improve the computational costs of solving the optimization problem that lies at the core of the EMS and to implement the optimization problem in C++ as a "first draft" on its way towards a final product. Thereby, an already existing EMS prototype on the basis of Matlab/GAMS embedded in a simulation environment modeling a hybrid power plant in Cobija (Bolivia) is utilized. This prototype is translated into the open source modelling environment Julia/JuMP. The solver used throughout most of the project is the commercial mixed-integer linear problem (MILP) solver CPLEX. However, it is shown that the open-source solver CBC can be a serious alternative, especially for smaller power plants than the example case Cobija.

Tests are conducted with the aim to qualitatively evaluate ways to keep the computation time of the solver within boundaries, while at the same time maintaining the quality of the results. As a measure of the EMS performance, the accumulated objective values over one day are compared. The conducted tests show, that setting a time limit to the solver, or extending the time to solve the problem, are good measures to keep the computation time within boundaries. The negative effects on the results are less pronounced than expected.

Furthermore, the optimization problem is implemented in the CPLEX solver specific application programming interface (API) in C++. The new C++ prototype is able to reproduce the expected results. In a future project, the input and output data processing will need to be implemented and refined.

Finally, an accumulated cost analysis is conducted that is able to show the economic utility of the EMS for a power plant operator.

Summary

REVIEW	3
SUMMARY	4
1. GLOSSARY	7
2. INTRODUCTION	8
2.1. Motivation.....	8
2.2. About GreenPowerMonitor	9
2.3. Previous requirements	10
2.4. Objectives and scope of the project.....	11
3. PROJECT FOUNDATIONS	14
3.1. The EMS project	14
3.1.1. Type of system.....	15
3.1.2. Stochastic model of uncertainty	16
3.1.3. System stability	18
3.1.4. Mathematical model of the system.....	22
3.1.5. Existing scripts and the Simulink model	25
3.2. The theory behind solving MILPs.....	29
3.2.1. Branch and bound algorithm	29
3.2.2. Cutting plane method.....	31
3.2.3. Other extensions and tools	32
3.3. The underlying theory of solving LPs.....	33
3.3.1. Primal simplex method.....	34
3.3.2. Dual simplex method	37
3.4. Deployed modelling tools.....	39
3.4.1. JuMP modeling language for mathematical optimization	39
3.4.2. CPLEX Concert technology – CPLEX’s C++ API.....	40
4. DECISION ON SOLVER	42
4.1. Solver Tests	42
4.2. License considerations	44
5. EMS IMPLEMENTATION IN JUMP	46
5.1. Implementation of the mathematical model	48
5.2. Embedding the Julia script.....	51
5.2.1. Embedding the OP for simulations.....	51
5.2.2. Embedding the OP for emulations	54



6. EXAMINATIONS FOR THE FINAL EMS IMPLEMENTATION	56
6.1. Adapting the OP execution period.....	58
6.2. Enforcing the OP execution period.....	60
6.2.1. Setting a time limit in the CPLEX optimizer	60
6.2.2. Using previous results	61
6.2.3. Discussion and comparison	61
6.3. Reducing the number of scenarios.....	62
6.4. Warm start / solver change.....	64
6.5. Modelling the problem as a LP instead of a MILP	66
6.6. Conclusions	67
7. EMS IMPLEMENTATION IN C++	70
7.1. Core EMS and testing environment	71
7.2. Input and output data format	73
7.3. Visual Studio settings	76
7.4. Program structure.....	77
7.4.1. LoadForecasts.cpp	77
7.4.2. Other external source files	78
7.4.3. Main function.....	79
7.5. Test.....	82
8. COSTS OF THE PROJECT	84
CONCLUSIONS	88
APPENDIX	96



1. Glossary

EMS	Energy Management System
OP	Optimization Problem
PV	Photovoltaic
LP	Linear Program/Problem
MILP	Mixed-Integer Linear Program/Problem
API	Application Programming Interface
VS	Microsoft Visual Studio
xml	Extensible Markup Language
csv	Comma-Separated Values
PPC	Power Plant Controller
GPM	GreenPowerMonitor
UPC	Universitat Politècnica de Catalunya

2. Introduction

The project presented in this master thesis is part of a bigger project to develop an energy management system (EMS) for the optimal operation of hybrid power plant in islanded power grids that contain a set of grid stabilizing diesel generators, a photovoltaic (PV) generator, and a storage system. The overlying project is initiated and supervised by GreenPowerMonitor (GPM) in Barcelona as a R&D project for the company. In the following, the reader is introduced into different aspects of this thesis project including its objectives.

2.1. Motivation

Due to the falling prices of photovoltaic (PV) generators and batteries, many islanded power systems all over the world that have been powered by diesel generators, are upgraded with, either only PV panels, or a combination of PV and a battery storage in order to save fuel costs. Introducing these additional components adds degrees of freedom to the control of the plant that carry the potential to optimize the power production in a desired way. The most obvious way, from an economic perspective, is to use the PV power in an optimal manner, by smartly controlling the battery storage, to save fuel. This smartness requires so-called energy management systems (EMS) that take the decisions on the behavior of the different components relevant to the objective of the system. This means, that the EMS is the highest technical instance in the system, coordinating the controllers of the power plant by manipulating their setpoints in a way that optimally fulfills the EMS' (plant operator's) objectives. GPM, as a manufacturer of monitoring and controlling solutions for large PV installations, has a natural interest in developing such a system, to integrate it with their controllers in order to satisfy customers with hybrid PV power plants. In a broader sense, all kinds of combined power generation systems need an EMS, however this thesis project focusses on the case with diesel generators. Some of the findings are however transferrable.

At the core of the EMS, a mathematical optimization problem has to be solved. This problem is what mainly builds up the EMS model and it offers many opportunities for innovation. Not only at the mathematical core of the problem formulation, but also at the technical aspects, like solver and hardware choice, parameter specifications, data handling, etc.

To explore these design questions of the optimization problem and to develop the EMS towards a product for GPM is what motivates the master thesis project on hand.



2.2. About GreenPowerMonitor

Remark: A slightly extended version of this section was already formulated by the student for his internship report. The text has been reused here.

GreenPowerMonitor (GPM) is a middle-sized software company located in Barcelona with small local branches abroad (e.g. in the UK). The company provides solar monitoring, -control and asset management solutions and is a leading player in that field. In summer 2016, GPM was acquired by the Norwegian technology corporation and classification society DNVGL and profits therefore from the global network that DNVGL has. At the moment GPM monitors roughly 5GW of solar power worldwide. Most of this solar capacity is installed in the US, Europe and Latin America (see Figure 1).



Figure 1: Activity of GPM worldwide in MW monitored

The company's three main monitoring solutions are

- PV SCADA
- PV+
- PV Portal

PV SCADA is the full solution for large scale photovoltaic plants with a server installed on site and real-time data measurement (1 second time resolution). PV+ and PV-Portal are based on

the GPM server in Zona Franca - Barcelona. PV Portal is the standard browser based monitoring solution while PV+ is a desktop application with additional functionality for customers with a large portfolio of power plants.

Additionally, GPM provides high level customer service based on the concept of key account managers that are the personal contacts for PV SCADA customers. Also, the hardware services and setting up the monitoring infrastructure is entirely covered by the GPM project management team.

Finally, GPM has developed a control algorithm for photovoltaic converters called GPM Power Plant Controller (PPC) that manages the power plant's parameters like active power, reactive power, frequency, voltage and also external reactive power compensation. This control keeps the real values of the parameters selected by the customer at their set points at all times.

To complement this portfolio, GPM is developing the EMS project, in order to be able to optimize the PPC operation for hybrid PV power plants.

2.3. Previous requirements

The first EMS prototype was developed at the CITCEA research institute at Universitat Politècnica de Catalunya (UPC) on the basis of Matlab/Simulink and GAMS. The work done at CITCEA is structured into 5 project phases:

1. Project definition
2. Definition of requirements and base case
3. Management algorithm development
4. Testing of algorithms in simulations
5. Testing of algorithms in emulations with real power converters

The first 4 stages were completed before the start of this thesis project and could therefore be considered when defining the objectives of the project. The 5th stage was running in parallel to this project and is only considered by implementing the functionality in the project code to be able to run emulations (see section 5.2.2).

The basic design decisions for the EMS are adopted for this thesis, and the Simulink model developed for phase 4 is used to test different designs of the EMS. For code in Julia or C++ developed in this thesis project, the previous versions in other environments could be used as example.



2.4. Objectives and scope of the project

The objectives of the project build upon the previous development steps of the EMS described above. After having a first prototype of the EMS, a need arose to explore different properties and behaviors of the system through tests. Thereby, it was focused on how the EMS results are affected by the limited time frame, and which measures are the most potent to tackle possible timing issues. The tests are conducted in the Simulink model of an exemplary hybrid power plant in Cobija (Bolivia) developed at CITCEA. Three different PV generation profiles are used. Then the results are qualitatively compared using the objective value of the results, and the time to solve the problem.

However, the optimization problem (OP) of the EMS was implemented in the commercial modelling environment GAMS, making it necessary to purchase a license for it. The lack of an academic license for the EMS problem size made it completely impossible to run these tests at GPM with the original prototype. Therefore, a first objective was to translate the project's optimization problem to an open source environment.

As a first step of developing an EMS product from the prototype, another objective was defined as to implement the EMS in the programming language preferred within GPM – C++. This can be seen as a first draft, from which later, the project will be taken further and more functionality will be added. Because of the somewhat different advantages of a neutral modelling environment and a solver specific C++ interface, these two objectives were faced separately in two steps (see chapters 5 and 7).

As a last objective, the economic potential of the EMS was to be evaluated. In the last chapter of this thesis, an accumulated cost analysis is conducted in order to estimate how feasible the EMS in the current layout is.

In short, the objectives of this master thesis project are summed up to:

- Transfer the GAMS implementation of the OP to an open source environment
- Find out about the dependencies of the EMS on different parameters in tests to facilitate future product development.
- Translate the project to C++
- Study its feasibility

In order to clarify the project scope, in the following some aspects are presented that do not belong to this thesis, and are left for later stages:

- Tests with real life equipment were already conducted at CITCEA. Further test will be necessary, once the EMS is getting implemented.
- A final version of the EMS. Final decisions about the solver, the hardware, different parameters, or designs will be only possible when the system is implemented on a real plant. The findings from the tests conducted in this thesis are meant to speed up the process of determining the final design for every plant. However, there will still be a learning curve, once the EMS implemented in a real-life application.
- This thesis focusses on the optimization problem of the EMS and how to practically implement it. Other system parts like forecast processing and the scenarios are not studied or implemented. The forecasts and scenarios used are identical with the once used and developed at CITCEA for the first prototype. Neither is the mathematical formulation of the OP changed. Changes are suggested based on test results, however the decisions, whether these changes are feasible or not, are left for further evaluation by a mathematician.

Concluding, this thesis qualitatively examines different designs of the EMS OP implementation and the effects these design decisions have on the EMS performance, and is meant to set the first brick on the way to a final product.



3. Project foundations

The project of this master thesis cannot be seen as an isolated work but rather as a contribution to the development of an energy management system at GPM. At the beginning of this thesis project, several development steps for the EMS had already preceded in the months before. This thesis builds upon the results of these earlier stages. Furthermore, to understand the implementations and decisions made during the course of this project, some basic knowledge about mathematical optimization and its computational implementation tools is required. Therefore, in the following sections, the EMS project stages until the start of this master thesis project are illustrated, a summary of the mathematical foundations of solving mixed-integer linear problems (MILPs) (and as a basis for that, linear problems (LPs)) is given, and finally, the auxiliary software tools used in this thesis are described.

3.1. The EMS project

Remark: The work presented in this section was already existent when starting the thesis project and the thesis builds upon that work. It is, however, not possible to cite publications, because the reports are created exclusively for GPM and are confidential. So far, there exist no publications on the described project phases. Responsible for the work presented in this section are the following members of CITCEA research institute: Andreu Vidal, Monica Aragüés, Eduard Bullich, Guillem Vinyals, and Oriol Gomis.

The energy management system (EMS) has the task to coordinate all the components in a hybrid PV power plant in such a way, that energy losses and fuel consumption are minimized. This is done by repeatedly solving a mathematical optimization problem (OP) in which the mentioned objective and all the physical constraints are mathematically modelled. As a result, the computation returns all the computed decision variables for which the objective function takes its maximum value. These variables are sent as setpoints to the control systems of the different components of the plant (battery storage, diesel generators, and PV converters) that than keep the real values close to these optimal set points. The basic principle is shown in Figure 2. The EMS takes into account forecast data for PV generation and consumption, and the actual state of charge of the battery. After solving the optimization problem, it sends power set points to the battery storage and the PV converter(s), and a number of diesel generators that need to be running. This means that the most important prerequisite for the functioning of the system is the existence of reliable forecast data and preferably much historical data for PV generation and consumption in order to generate generation and consumption scenarios (see section 3.1.2). Another crucial prerequisite is the existence on preferably powerful power plant controls and models for their behavior (see section 3.1.3) on which the EMS working principle



is based.

The steps of developing the EMS depend on the system the EMS is designed for. For every system, the steps in section 3.1.2 and section 3.1.3 have to be repeated to design the suiting EMS. The development steps of the EMS are summarized in the following. At the end of this section, the scripts and Simulink models developed before the start of this thesis project are presented in short. Some of the basic theory of this section can be studied more in detail in [1] and [2].

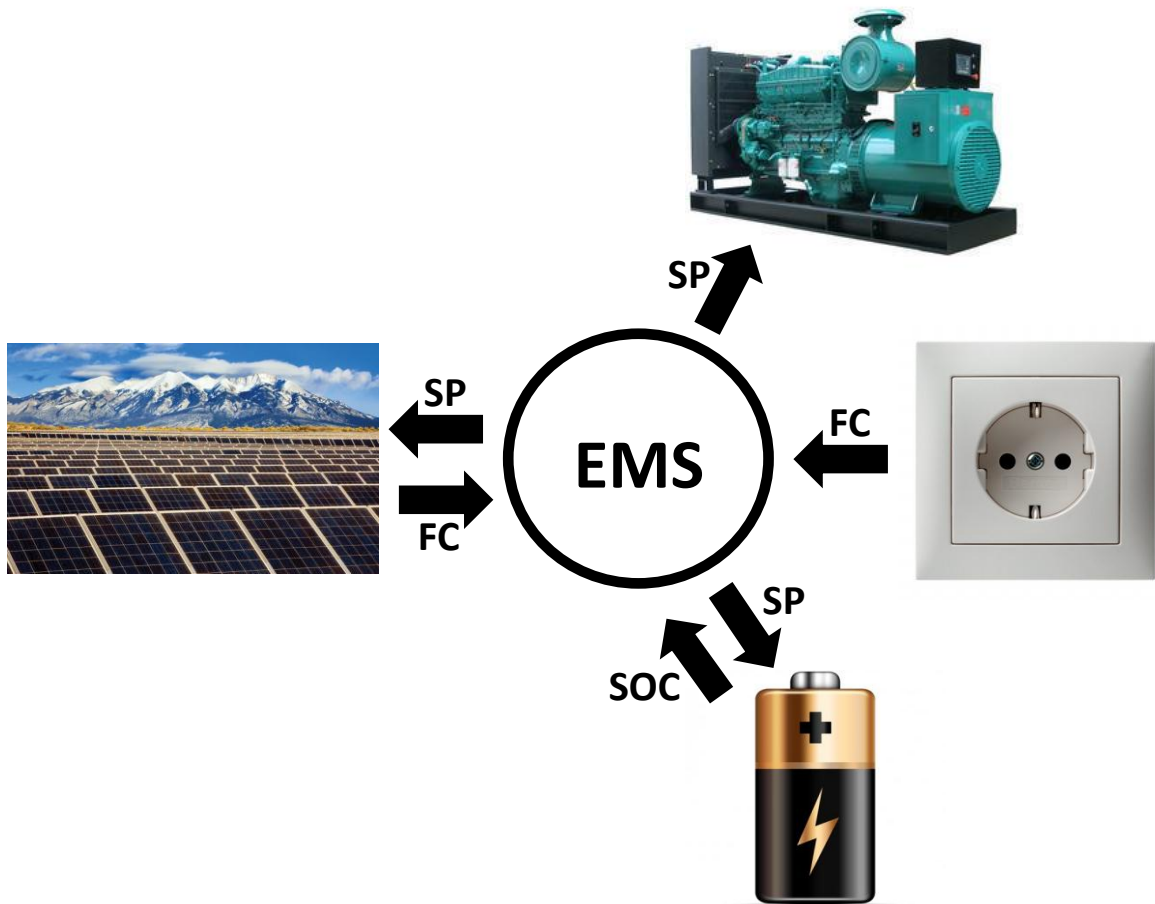


Figure 2: Working principle of the EMS (SP=set points, FC=forecasts, SOC=state-of-charge)

3.1.1. Type of system

The first step is to determine which kind of power plant the EMS will be designed for. The factors are:

- Is the system connected to a grid or does it work in an islanded mode?

- How many diesel generators are installed in the system?
- Does the system have a battery?
- What quality do the available forecasts have? Are cloud movements observed (short term PV generation forecasts)?
- What is the relationship between installed PV power, consumption, and available diesel power?

All these factors influence in what order of magnitude the power deviations are, how significant the frequency deviations are allowed to be, and therefore, how much power reserve is required (diesel generators that are always running with a power margin).

In the case of the EMS project, a power plant in Cobija (Bolivia) was used as a test case. The properties of this power plant are listed in the following:

- The system works islanded, that is, not connected to a superior grid
- The installed PV power almost never exceeds the consumption. That means, that the PV power can be seen as support of the diesel generation which takes the main role
- The diesel generators maintain the system frequency. In every moment, there is inertia in the system.
- The battery storage works as frequency support regarding variations in PV generation and consumption
- The battery storage does not work as long or medium-term storage
- The minimum diesel power allowed, compromises the maximal usable PV power
- The PV generation shows a high degree of variability, especially in the afternoons
- The measured consumption in the point of common coupling shows clear patters even though sometimes there are deviations. This facilitates the prevision of the consumption.
- All the system components can be regarded as connected in the same point in the system which means that effects on system voltage/reactive power can be disregarded.

3.1.2. Stochastic model of uncertainty

In order for the EMS to correctly minimize battery losses and fuel consumption, it depends on high quality statistic input data. Therefore, it is necessary to find the right forecast models



based on long term observations.

The class of model that is needed for the EMS is called “two-stage problem”. The principle of these kind of problems is, that a decision is taken based on the known parameters at the time of the decision. This decision may have different possible outcomes in the future (second stage), on which the system responds with another decision based on a decision catalogue. This second decision can compensate for possible negative effects of the first decision. Different future outcomes for the second stage decision are modelled as scenarios that are obtained by analyzing the statistical distribution of historical consumption and PV generation data, also called time series. If we, for example, look at the statistical distribution of the consumption in Cobija (Bolivia), we observe that the data is normal distributed. Thus, the mean value of the distribution and the standard deviation can be calculated and scenarios can be randomly generated based on these mathematical parameters.

With this approach, the uncertainty of calculating optimal set points for future time periods is included into the mathematical problem formulation. In case of this project, parameters of the first stage are the number of running diesel generators and the maximum allowed PV power. These variables are identical for every scenario, because once a decision is made, these parameters will not change. Second stage variables are diesel generator output power and battery power, since the controls of these components will react on frequency changes that are the results of the first stage decision. These variables are calculated separately for every scenario. That means that adding scenarios to the model, significantly increases its complexity since every scenario adds variables and constraints to it. Therefore, the number of scenarios must be limited to those scenarios with significantly high variability.

It is important to say, that the availability and quality of historical data is crucial for the efficiency of the EMS. In case of this research project, time series data of about one month (only labor days) has been deployed to obtain the statistical parameters and distribution of consumption and generation. If there was data of one or several years, the model could also include season specific patterns and thus, the EMS would be even more efficient because the statistical distribution parameters could be chosen less conservative.

The stochastic model of the plant is composed of two model parts:

- The battery model (or long-term model) with a time horizon of an entire day that makes the EMS manage the SOC of the battery in an optimal way. That means, for example, that the preferred SOC around noon is very low, so that preferably much solar power can be used at a later point in time and thus, decreasing the fuel consumption. When there is no PV generation, the SOC should be preferably high to support the diesel generation in times of high consumption.

- The EMS model (or short-term model) with a time horizon of several minutes, that makes the EMS manage all the other system components in an optimal way.

These two parts have to be considered when generating scenarios for the EMS. In the course of the project, the historical data (several days for the long-term model, and 48 hours for the short-term model) have yielded the following distributions:

Long term		Short term	
PV Generation	Consumption	PV Generation	Consumption
Beta	Normal	Normal, parameters depend on quality of cloud forecast	Normal

Table 1: Statistical distribution of historical data

By means of this information and the obtained distribution function parameters, several scenarios are generated for PV generation and consumption that can be deployed in the optimization problem of the EMS. In the following, the schematic development of the statistical models and the scenario generation from these is summarized:

1. Examination of historical data for consumption and PV generation in regard to distribution patterns. This analysis should be executed several times for different seasons and different types of days (Labor Day, Saturday, holidays). The data is examined on a long-term and a short-term time scale.
2. Saving the distribution parameters for the site.
3. During EMS operation, the distribution parameters are used to create consumption and PV generation scenarios by randomly deviating the available forecast data with the distribution functions determined before.

3.1.3. System stability

As mentioned in the introduction of section 3.1, an important prerequisite for the operation of the EMS is to have a model of the power plant and its PPCs. This model is used for three purposes in the project:

- Find out the worst case of power deviations for the EMS to consider and its effect on the system frequency



- Find out the impact of the three variables that the EMS can manipulate on the system frequency: number of diesel generators that are turned on, battery power, and PV power limit
- Use the power plant model for simulations to test the performance of the EMS after completed development

Therefore, an important step in the EMS project was to develop a Simulink model of the Cobija power plant. In this section only main characteristics of the model from the view of the EMS are summarized.

The diesel generators' output power is controlled by a PI-control that keeps the system frequency at 50Hz. For the EMS, the diesel output power is variable. Thus, the diesel power can be seen as a slack variable, balancing out power fluctuations. In tests it has been identified, that the controller's step response has a dip and an overshoot in frequency, whereas the dip is significantly more pronounced than the overshoot. Thus, the critical limit used for the EMS is the minimum allowed frequency.

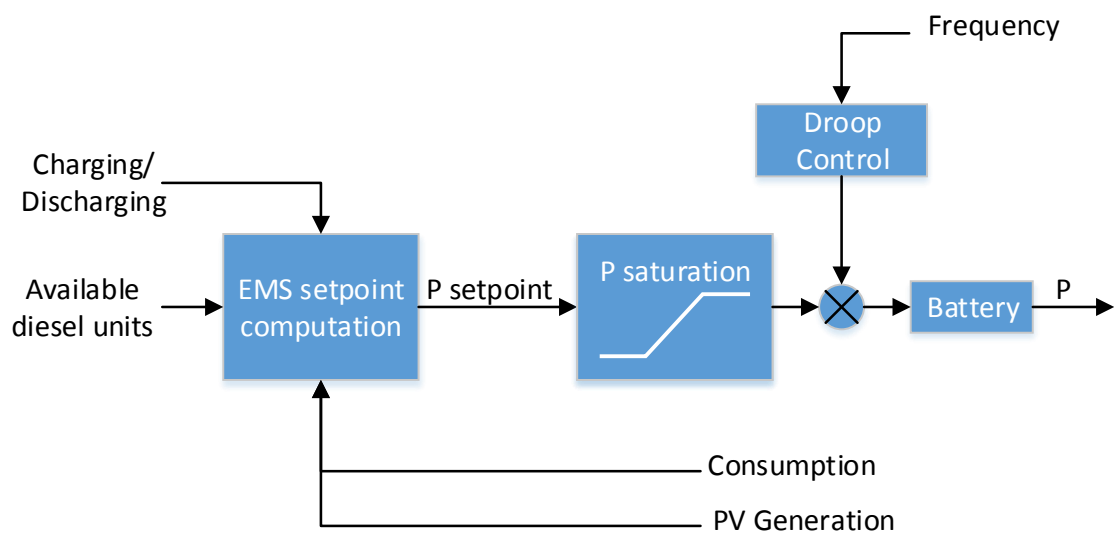


Figure 3: EMS battery control

Moreover, the PV generator and the battery contribute to frequency stability by means of a droop control. The underlying principle is that the control follows a droop curve that assigns a certain change in output power to a certain change in system frequency. This simulates the behavior of a synchronous generator and prevents the converters from “fighting” for control, meaning starting to oscillate heavily as response to frequency deviations. The battery has an energy reserve of 10% of the total SOC implemented to be able to always contribute to system stability. This frequency control is an inner loop with significantly finer time resolution than the

EMS loop. That means that the battery's output/input power is determined by the EMS for every 30-second period after which the droop control of the battery can modify the power in an inner loop to contribute to system stability. The schematic battery control is displayed in Figure 3.

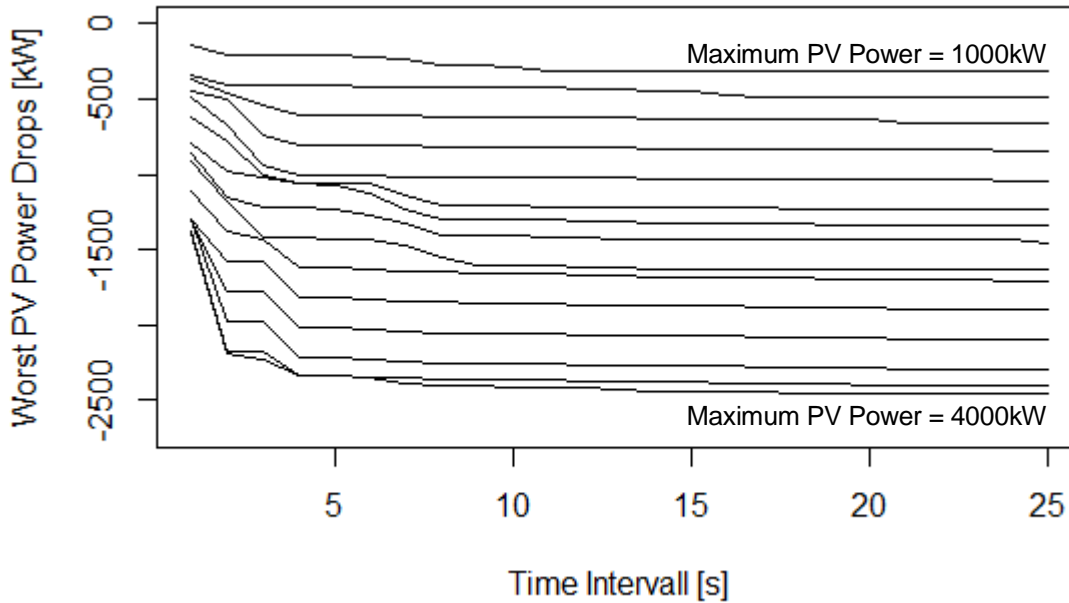


Figure 4: Worst negative power deviations depending on maximum PV power (top line: 1000 kW, bottom line 4000 kW)

The worst case of power deviations for the EMS to consider is composed of largest increase in consumption and largest decrease in generation. To obtain the worst case in consumption, the simulation data (time resolution: 1 second) was scanned for the most pronounced steps between two seconds. Then this was repeated for a time period of 2 seconds and so on up to a time period of one minute. The results would look like one of the lines in Figure 4. Taking into account the fact, that the examined data is not totally representative for every single situation, the worst cases for every time resolution are multiplied with a security factor. For the generation, this procedure is complicated by the fact, that the largest deviations depend on the available solar power. When there is a high solar generation, the power drops will be higher than if there is only little available solar power. Therefore, the approach for the consumption is repeated iteratively for different available solar powers. The results for exemplary generation data are plotted in Figure 4. The two cases are then combined to obtain the worst frequency deviations for the EMS to consider. In case of this project, this was done by simply adding up both cases which is the most conservative approach. A less conservative way of combining both cases would be to obtain a simultaneity factor. This would have a positive impact on the PV power curtailment, the necessary energy reserve of the battery, and the minimum number



of diesel generators that need to be running at all times.

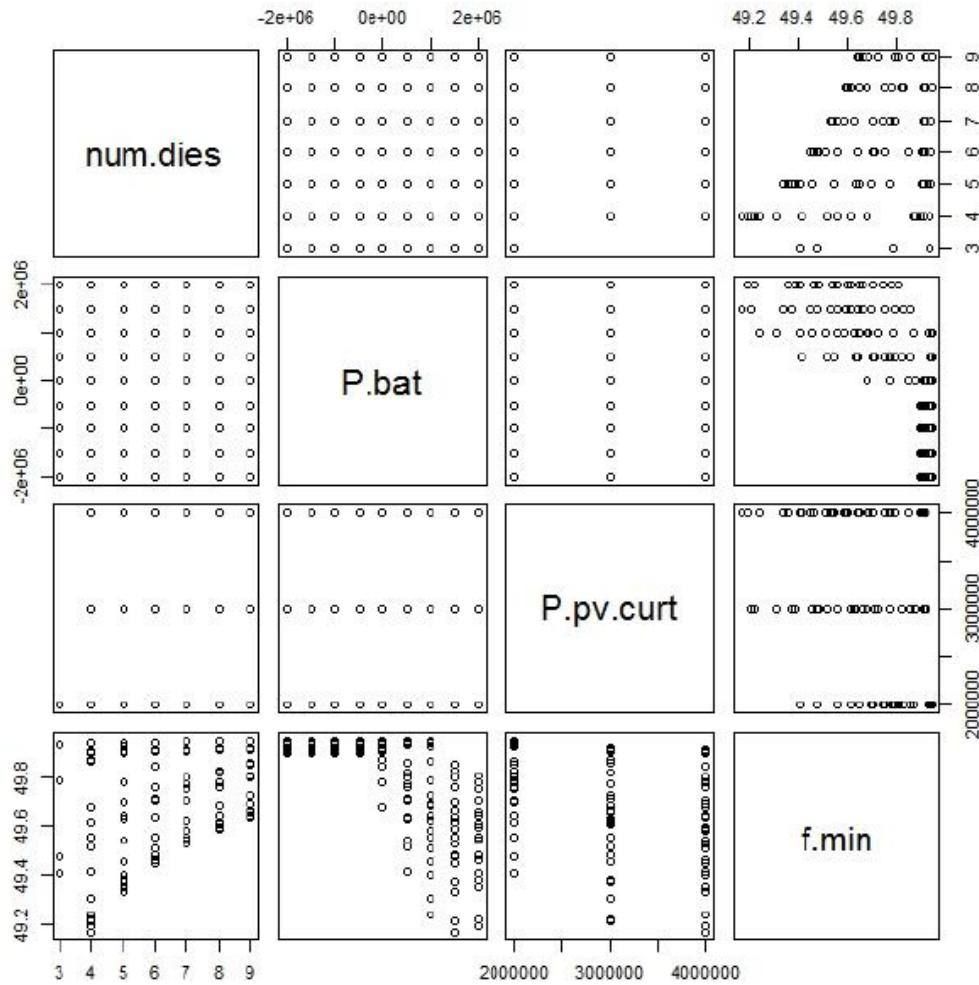


Figure 5: Result of multi-regression for the frequency constraint

The worst case of power deviations found out in the foregoing section is now used to obtain the system stability constraint, included in the mathematical model of the EMS. This inequality constraint relates the three variables over which the EMS has control (number of running diesel generators, PV power limit, and battery power) to the system frequency and defines this system frequency to be higher or equal than the minimum allowed system frequency. This inequality constraint is formulated as follows, assuming linearity:

$$f_{min} \leq \Phi_0 + \Phi_1 * n_{DieselON} + \Phi_2 * P_{bat} + \Phi_3 * P_{PVmax} \quad (1)$$

with f_{min} being the minimum allowed system frequency, $n_{DieselON}$ being the number of diesel generators that are running (can be modified to be the number of diesel generators running over which the EMS has control, that is without the minimum number of diesel generators), P_{bat} being the battery power, and P_{PVmax} being the PV power limit. The coefficients Φ_0 to Φ_3

are determined experimentally using the Simulink model of the power plant for which the EMS is developed (in case of this project: the Cobija power plant). A series of load and generation steps are applied to the Simulink model modifying $n_{DieselON}$, P_{bat} , and P_{PVmax} . The step size is the worst case for consumption and PV generation obtained in the previous part. For every step simulation, the minimum frequency is registered, and then, a multiple linear regression is applied to find the coefficients of the stability constraint (1). The results of the step simulations are plotted in Figure 5. The three variables adjustable by the EMS (number of connected diesel generators, battery power, and PV power limit) are displayed in relation to the minimum measured system frequency after the step occurred. The important graphs are the three on the right-hand side and on the bottom of Figure 5 respectively.

It can be noted, that for negative battery power (battery is charging), the relation to the system frequency is not linear. Therefore, only the positive battery power is included in the frequency constraint of the mathematical model of the EMS. This is sufficient, because only deviations of the positive battery power cause critical frequency deviations.

3.1.4. Mathematical model of the system

The type of the system defines the mathematical problem that has to be solved by the EMS. Properties of the mathematical model in case of this project are:

- The constraints introduced by the battery and PV models are a set of linear equations.
- The power balance equations and the variable bounds are a series of linear relations.
- Diesel generator state (ON/OFF) introduces binary variables into the model. This is where the model changes from LP to MILP which significantly increases the computational effort to solve the problem (see sections 3.2 and 3.3).
- In some cases, to correctly model the PPC behavior, it can be necessary to add conditional constraints which also add binary variables to the model.

In case of this project, neither the battery wear, nor the dependency of diesel generators on the rotational speed has been modelled. These factors can introduce non-linearities into the model. Therefore, in case of this project, the system model belongs to the OP model class Mixed-Integer linear problem (MILP).

Formulating the mathematical model for the EMS means connecting all the theory and development steps discussed in the previous parts of section 3.1. It is the core of the EMS, meaning that it contains all the theory behind it in formulas. Other parts of the EMS, apart from



the mathematical model, are the scenario generation, programming theory when implementing the model, and underlying technology like the power plant controller which is necessary for the EMS to work.

In the following, the indices (sets), decision variables, input parameters, and the objective function are introduced for a better understanding of later chapters in this work. Not all the variables are shown here and most of the constraints of the model are kept confidential in this thesis, since the EMS is still under development at GPM.

3.1.4.1. Indices

There are four sets in the mathematical model of the EMS which are used to calculate indices and counters:

Scenarios to consider for the stochastic optimization

$$S = \{1, \dots, n_S\}$$

Number of executions of the optimization problem left until 00:00

$$T^{EMS} = \{1, \dots, n_{T^{EMS}}\}$$

Number of 30 second periods between two executions of the optimization problem

$$T^{intra} = \{1, \dots, n_{T^{intra}}\}$$

Number of diesel generators

$$N^{Diesel} = \{1, \dots, n_{Diesel}\}$$

3.1.4.2. Decision Variables

In the following, the most important decision variables of the EMS for the understanding of this thesis are displayed:

Battery power every 30 seconds in kW

$$P_{t,p}^{bat}, t \in T_{EMS}, p \in T_{intra}$$

Binary variable for exclusion of simultaneous charging and discharging

$$X_{t,p}^{car} \in \{0,1\}, t \in T_{EMS}, p \in T_{intra}$$

State of charge of the battery every 30 seconds in p.u.

$$SOC_{t,p}^{bat}, t \in T_{EMS}, p \in T_{intra}$$

Diesel power every 30 seconds for every scenario in kW

$$P_{t,p,s,d}^{dies}, t \in T_{EMS}, p \in T_{intra}, s \in S, d \in N_{diesel}$$

Binary status of all diesel generator units (ON(1) / OFF(0)) every OP execution period

$$ON_{t,d}^{dies} \in \{0,1\}, t \in T_{EMS}, d \in N_{diesel}$$

PV power every 30 seconds for every scenario in kW

$$P_{t,p,s}^{PV}, t \in T_{EMS}, p \in T_{intra}, s \in S$$

PV power limit every 30 seconds in kW

$$P_{t,p}^{PVmax}, t \in T_{EMS}, p \in T_{intra}$$

These decision variables (among others) are returned as a result after the OP execution together with the maximum possible value of the objective function (see below). All these variables are necessary for the mathematical optimization. However, the EMS only sends the results of $\{P_{1,1}^{bat}, \dots, P_{1,p}^{bat}\}$, $\{P_{1,1}^{PVmax}, \dots, P_{1,p}^{PVmax}\}$, and $\sum_{n=1}^d ON_{1,d}^{dies}$ as setpoints to the power plant controller. These are the setpoints until the next EMS execution provides new setpoints.

3.1.4.3. Input Parameters

The EMS requires some input parameters before the computation can start. These parameters are listed in the following:

Consumption scenarios every 30 seconds in kW

$$L_{t,p,s}^C, t \in T_{EMS}, p \in T_{intra}, s \in S$$

PV generation scenarios every 30 seconds in kW

$$L_{t,p,s}^{PV}, t \in T_{EMS}, p \in T_{intra}, s \in S$$

*Battery capacity in kW*30s*

$$Cap^{bat}$$

State of charge of the battery when input parameters are dispatched in p.u., also called "initial



SOC" because it is the SOC value that foregoes all future SOC values in $SOC_{t,p}^{bat}$.

$$SOC^i$$

Battery efficiency in p.u.

$$\eta^{bat}$$

Power boundaries of the battery and a diesel generator unit in the generator set in kW

$$p^{mxB}, p^{mnB}, p^{mxD}, p^{mnD}$$

SOC boundaries for the battery in p.u.

$$SOC^{mx}, SOC^{mn}$$

Minimum allowed system frequency in Hz

$$f^{mn}$$

Required power margin for stability purposes as sum for the whole diesel generator set in kW

$$marge^{dies}$$

3.1.4.4. Objective Function

The EMS is formulated as a maximization problem. The objective function of the EMS model maximizes the total utilized PV energy over 24 hours while minimizing the battery losses over the same time interval. Furthermore, this energy is summed up for every considered scenario. This way, the result takes into account the uncertainty of generation and consumption forecast data and makes the setpoints calculated by the EMS valuable for many different futures. The objective function is formulated as

$$\text{maximize} \sum_{t,p,s} (P_{t,p,s}^{PV} - n_s * (1 - \eta^{bat}) * \text{abs}(P_{t,p}^{bat})) \quad (2)$$

3.1.5. Existing scripts and the Simulink model

Remark: A slightly adjusted version of this section was already formulated by the student for his internship report. The text has been reused here.

To execute the tasks described in the previous sections (3.1.1 to 3.1.4) and to run Simulations to test different versions of the EMS without a power plant (or at least physical converters), different R-scripts and a Matlab Simulink simulation environment were created during the

course of the EMS project. In the following, these scripts and the Simulink model are presented shortly.

The statistical part, described in sections 3.1.2 and 3.1.3, is processed by means of programs written in R, a language for statistical computing, stored in the folders “00 - DADES SEGUNDALS” and “1 - PROBLEMA DETERMINAR SOC”. To run the scripts, it is necessary to download the R-environment and the software R-Studio to edit or run the scripts. Moreover, the scripts require certain packages. These packages will be displayed in an error message in the console window when trying to run a script. To download and install the necessary packages the user can type `install.packages("packagename")` into the console and press enter. Normally the package will be downloaded and unpacked automatically then. In the following the R-scripts are listed:

- The script “**1 - Agreggated _data.R**” does two tasks of reading raw data: First the DIA 1 data is loaded and consumption and PV generation is saved as “consum_segundal.RData” and “generacio_segundal.RData”. These files are the basis for entire day simulations with HEMS (see above). Then the data from DIA 1 and DIA 2 (resolution 1sec, only from 8am-8pm) is loaded and stored as a mat-file “M_Data.mat”. This file is used in the Load-Step Simulation Series
- “**Agragated_data_noves_dades.R**” is used in addition to “Agreggated _data.R” when alternative generation data is used. The user sets the data number, e.g. “3” to read the generation data in the folder “Dia 3” also used in this thesis (later called “Data 3”).
- “**2 - Time_series_Analisis.R**” reads the data in the folder “TS_ANALYSIS” and creates an ARIMA model in order to be able to make forecast emulations of the consumption to use in simulations and to create scenarios from forecast data for every new day.
- “**Var_otg_Resta_del_dia.R**” creates a vector of consumption forecasts for the day that is going to be simulated and stores it as “forecast_cons_final.mat”.
- “**0 - Model variabilitat Previsions_corregit.R**” generates a prevision and a model of the statistical errors of the available solar power. The results are saved as “forecast_gen.mat”.
- “**0 - Positive_changes.R**” and “**0 - Negative_changes.R**” determine the most significant generation/load changes for different time intervals (1,2,3,4,... seconds) and store them as M_change_neg.mat and M_change_pos.mat respectively in the sub-directory “Fitxers_resultats”.



- “**Generacio Escenaris corregit.R**” creates the different scenarios for the PV generation using the results of the previous parts. These scenarios are then used for the optimization during the simulation. Historically also the consumption scenarios were generated using this script. Now however, they are generated during the EMS execution. This way scenarios for both consumption and generation will be created in the final implementation of the EMS.

In the following, the Simulink simulation environment developed at CITCEA is introduced. In this environment, the power plant in Cobija, which is used exemplarily in this work to develop the EMS, is modelled with all its components. These include

- Physical models of PV generator (panels and inverter), battery system (battery and converter), and diesel generator set.
- Droop controls of PV generator and battery system for the simulation of the behavior of a synchronous machine.
- Centralized controller for every component as part of the PPC, including diesel generator set, battery system, and PV generator.
- Power balance model of the plant, modelling the diesel generator power as slack variable of the system: $P_{consumption} - P_{battery} - P_{PV} = P_{diesel}$.
- “Input sockets” for the simulation parameters and simulation input data (e.g. consumption). Allows the user to choose between simulation data input as Matlab vectors in the workspace, or step functions for the stability examinations of the system (see section 3.1.3).

To perform the step simulations for the frequency constraint coefficients (see section 3.1.3), the user first executes the script “parameters_balanc.m” in the folder “0 – SIMULINK”. This script sets all the parameters used by the Simulink model in the Matlab base workspace. Then the script “LOOP_simulations_HEMS_v1.m” is able to perform a series of load- and generation step simulations with value ranges for PV power, battery power, and number of diesel value arrays specified by the user in the script. After the very short simulations have finished, the multi-regression to find the relationships of the adjustable value ranges and the system frequency response, can be executed with the R-script “read_results.R” in the same folder and the results are plotted as R-plots.

The same simulation environment, simulating the Cobija power plant, is used to test the EMS in simulations with the length of one day. However, the EMS model is added which is an embedded Matlab function, triggered every OP execution period (e.g. every 5 minutes of

simulation time), requiring the current simulation time and battery SOC as input parameters and returning the setpoint arrays containing the values until the next EMS executions for all the EMS controlled elements in the model. The EMS Simulink model is shown in Figure 6. Furthermore, instead of step functions, the simulation model for EMS tests uses generation and consumption input data over one day with time resolution of one second. These simulation time lines are generated with the scripts “1 - Agreggated _data.R” and “Aragated _data _noves _dades.R”. The Simulink file containing the EMS model in addition to all the other power plant components listed above is called “HEMS_v4.slx” in the folder “2 - MODEL MATEMATIC 60S”.

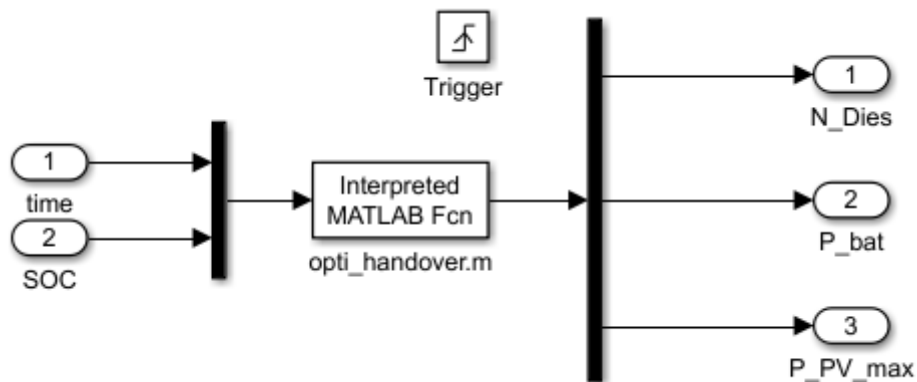


Figure 6: Simulink model of the EMS

Also for this Simulink model, it is necessary to run the script “parameters_balanc.m” first to initialize all the necessary simulation parameters in the Matlab base workspace. However, a superscript called “Execute.m” was created by the student before the beginning of this thesis project, that automates the entire process of running the single scripts and simulating the model for an EMS test. Depending on the user configuration, it runs the statistical scripts to prepare new data for the simulation, runs the script “parameters_balanc.m”, simulates a day with the Simulink model, saves the simulations results from the base workspace and the optimization results that are buffered in the general script folder during the simulation, and plots the results for later analysis of the EMS performance. The plots are also saved automatically if required.



3.2. The theory behind solving MILPs

Mixed-integer linear problems in standard form are defined as

minimize

$$\mathbf{c}^T * \mathbf{x}$$

subject to

$$\mathbf{Ax} \leq \mathbf{b}, x_i \geq 0$$

with

(3)

$$\mathbf{A} \in \mathbb{R}^{p \times n}, \quad \mathbf{x} = (x_1, \dots, x_n), \quad \mathbf{c} = (c_1, \dots, c_n),$$

$$\mathbf{b} = (b_1, \dots, b_p) \text{ with } b_j \geq 0$$

and

$$x_i \in \mathbb{Z}, \quad i \in \{1, 2, \dots, n\}$$

The last row is called the integrality condition of the MILP and makes the difference between a LP and a MILP. It states, that some (or all) of the variables in the solution vector must be integer values. Some of these variables can also be restricted to be either 0 or 1, which makes them binary variables. More details on LPs and the standard representation are given in section 3.3.

State of the art solvers for MILPs nowadays use an approach that is often called branch and cut algorithm which is a combination of the traditional branch and bound algorithm to solve mixed-integer problems (not necessarily only linear), the cutting plane method, and other tools like heuristics. Cutting planes are a powerful tool to reduce the total size of the problem-to-solve. However, in modern MILP solvers, way more methods and extensions are exploited. In the following, a functional overview about the branch and bound algorithm, the cutting plane method, and some other additional methods in modern MILP solvers is given, based on the description in [3].

3.2.1. Branch and bound algorithm

The branch and bound algorithm is based on the idea to subdivide the original MILP into smaller problems and treat them as if they were normal LPs. That is the reason, why a MILP solver always uses an underlying LP solver to solve the sub problems.

The first step is to solve the original MILP as if it was a normal LP. That means, that the integrality conditions for the integer variables are dropped. This first problem relaxation is called root relaxation. If the integer variables of the optimal solution all have integer values despite not explicitly enforced, this solution is the optimal solution for the original problem. However, the likelihood that the root relaxation also solves the MILP is very low and normally the root relaxation result will contain integer variables with non-integer values. In that case, the next step is to select one of these variables and “branch” on it. That means that two sub-problems are created that have the additional constraint, that the selected variable has to be greater or equal and lower or equal than the neighboring integer values of the non-integer value of the “branching variable” respectively. Exemplary this step could look as in the following:

The root relaxation was solved, yielding an optimal solution that assigned the value 5.72 to one of the integer variables which is called “x”. Thus, it is possible to branch on this variable. The two sub-problems have the additional constraint $x \leq 5$ and $x \geq 6$ respectively.

After this step, one of the branches is chosen and again the LP relaxation is solved. This procedure goes on until one of the optimal solutions of an LP relaxation also fulfils the MILP, meaning that all its integer variables have integer values. This solution is also a solution to the original (less restricted) MILP and is called “feasible solution”. If the objective value of this feasible solution is better than all the other feasible solutions’ objective values, or if there were no feasible solutions yet, the actual feasible solution is called the “incumbent”. In any case this feasible solution marks an endpoint of the tree and can be fathomed. That means that it is not necessary to branch again on this variable. However, yielding a feasible solution is not the only reason for a branch to be marked as fathomed. Another reason is, if the current LP relaxation cannot be solved. If the LP relaxation is infeasible, then obviously, there is no feasible MILP solution either. A third reason is, if the optimal solution of the current LP relaxation has an objective value that is worse than the objective value of the current incumbent. In this case, it is impossible to obtain a better feasible MILP solution on the current branch and it becomes unnecessary to continue branching further down on it.

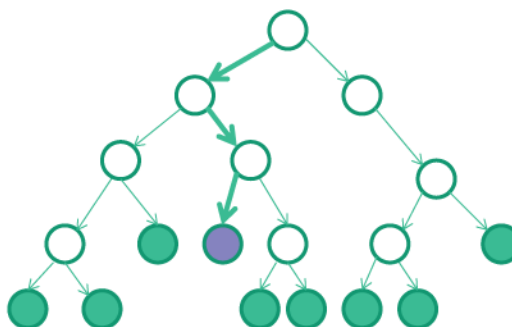
This algorithm generates a “search tree” in which every node is a sub-problem of the original MILP. Fathomed solutions are symbolized by leaves of the tree since they represent endpoints. The “leaf” with the best objective value is the final optimal solution of the original MILP.

The incumbent during the branch and bound execution can also be seen as an upper bound (in case of minimization) to the final optimal solution because the final solution will either be the current incumbent or one with an even better objective value. A lower bound (in case of minimization) is given by taking the minimum objective value of all the current nodes in the search tree. If the difference between the upper and the lower bound, called “gap”, is zero, the



current incumbent is the optimal solution to the MILP.

Branch-and-Bound



Each node in branch-and-bound is a new MIP

Figure 7: Schematic image of the branch and bound algorithm [3]

In principle, the schedule for sub-problems to branch on is chosen according to the objective values of the leaves. That is, all the objective values of the different available LP relaxations are compared and the LP subproblem with the best objective value is chosen.

In conclusion, the branch and bound algorithm makes MILPs solvable by using very well explored LP solving algorithms by increasing the total problem size which is also the major disadvantage of it. During the last years, the performance of the classic branch and bound algorithm could be drastically increased by adding numerous “tricks” that all have in common, that they reduce the size of the search tree part, that has to be explored. The most important improvement has been the cutting plane method that changed the name of the branch and bound algorithm to branch and cut algorithm.

3.2.2. Cutting plane method

The cutting plane method has been the most powerful improvement of the branch and bound algorithm in the past. To speed up the branch and bound algorithm, it is crucial to reduce the number of problems to solve, and cutting planes were found to be a powerful tool to do so. By introducing additional constraints, called “cuts”, to the LP relaxations dynamically, domains of the search tree that contain only fractional solutions are “cut off”. That means that the number of variables to branch on is reduced. Normally, many cutting planes are added already before branching for the first time, right after solving the root relaxation. In case of the biggest instance of the EMS OP, CPLEX adds tens of thousands of cuts to the root problem. These additional constraints considerably reduce the search tree size. However, also during the process of exploring the search tree, new cuts can be added to the subproblems to cut off branches, in cases when a cut possibility only became visible at a certain subproblem.

The subject of cutting planes is still being explored and special cutting planes can even mean a competitive advantage for a commercial solver. In the following, an example on one famous cutting plane is cited to illustrate the cutting plane method.

Suppose our formulation includes the following constraint: $6x_1 + 5x_2 + 7x_3 + 4x_4 + 5x_5 \leq 15$, where x_1 through x_5 are restricted to be binary. Suppose in addition that we have just solved an LP relaxation and that these variables take the following values in this LP relaxation: $x_1 = 0$, $x_2 = 1$, $x_3 = x_4 = x_5 = 3/4$. This undesirable solution can be excluded with the following observation: since $7 + 4 + 5 = 16 > 15$, it is not possible that $x_3 = x_4 = x_5 = 1$, and hence that the following new inequality is a valid addition to the given MIP: $x_3 + x_4 + x_5 \leq 2$. Since $3/4 + 3/4 + 3/4 = 9/4 > 2$, the new inequality cuts off the current solution. This inequality is an example of a so-called knapsack cover. [3]

More information about different cuts in CPLEX is available in [4].

3.2.3. Other extensions and tools

In the following, the most important additional tools to the branch and cut algorithm are introduced. Modern MILP solvers, however, come with way more techniques to speed up the search for the optimal solution. Many of which are held confidential to keep a competitive advantage.

The MILP **presolve**, reduces the initial problem size and tightens its formulation. This is done, for example, by identifying variables that only have one possible value they can take. These variables are part of the final solution and can be substituted in the whole problem formulation. The presolve is divided in MILP presolve and LP presolve, the former fixing integer variables at an integer value and being a way more powerful tool than the latter, that only fixes continuous variables. Also, eliminating redundant constraints or merging constraints is part of the presolve.

Heuristics are a powerful tool to obtain good feasible solutions to the problem sooner in the branch and cut process. A heuristic is a non-deterministic approach to the problem that takes advantage of experiences with different types of problems. Heuristics are very problem specific and can make a big difference between different MILP solvers. The reason for this is, that it is very valuable for the branch and bound process, when a good incumbent is found fast, because more relaxations will have worse objective values than the incumbent and can therefore be marked as fathomed. Furthermore, in cases when the solver has only limited time (as is the case for the EMS OP in this thesis), the faster the objective value of the incumbent improves, the better the final result of the optimization. There are numerous kinds of heuristics



and solvers of different vendors come with different heuristics. Usually, the experienced user is given flexibility to try out different heuristics for his problem himself in order to improve solve times. CPLEX provides the possibility to set the behavior in terms of applying heuristics during the solve process to “aggressive”, meaning that it spends more time on applying heuristics. An exemplary heuristic is cited in the following to illustrate the idea behind the topic:

[...] it has turned out to be extremely valuable to do a little extra work at some of the nodes of the search tree to see if a good integer feasible solution can be extracted, even though integrality has not yet resulted due to the branching. For example, it may be that many of the integer variables, while not integral, have values that are quite close to integral. We could then consider rounding some of these variables to their nearby values, fixing them to these values, solving the resulting LP relaxation, and repeating this procedure several times in the hopes that all integer variables will fall into line. If they do, and if the resulting feasible has a better objective value than the current incumbent, we can replace that incumbent and proceed. [3]

The natural **parallelism** that lies in the branch and bound procedure that is due to the numerous search tree leaves that can be solved independently, can be exploited very well when the solving computer has several CPUs or cores.

3.3. The underlying theory of solving LPs

As described in section 3.2, an important part of solving MILPs, is to solve linear relaxations of the basic problem and sub-problems. Therefore, the basis of solving MILPs fast, is to use a fast LP solver. In case of this work, the CPLEX optimizer was chosen to solve the EMS optimization problem (see chapter 4). In continuous mode CPLEX provides the following solving algorithms [4]:

- Primal simplex
- Dual simplex
- Networked simplex
- Barrier
- Sifting

In default mode, CPLEX solves the EMS problem by using the dual simplex algorithm. Therefore, in the next two sections, the theories of both the primal (as a theoretical basis) and the dual simplex methods are explained.

3.3.1. Primal simplex method

The primal simplex method is applied on linear optimization problems in the standard form:

maximize

$$\mathbf{c}^T * \mathbf{x}$$

subject to

$$\mathbf{Ax} \leq \mathbf{b}, x_i \geq 0 \quad (4)$$

with

$$\begin{aligned} \mathbf{A} \in \mathbb{R}^{p \times n}, \quad \mathbf{x} = (x_1, \dots, x_n), \quad \mathbf{c} = (c_1, \dots, c_n), \\ \mathbf{b} = (b_1, \dots, b_p) \text{ with } b_j \geq 0 \end{aligned}$$

Every linear program can be brought into standard form by means of simple mathematical operations. This linear problem is a convex polytope (possibly unbounded). The principle of the simplex method is to jump from one extreme point of the polytope to the next along the edges until the objective function cannot be improved anymore (see Figure 8). The extreme point of the polytope with the best objective value is the optimal solution to the problem. Extreme points are also called basic feasible solutions. If there is a basic feasible solution with a better objective value than the current one, there is also an edge leading away from the current basic feasible solution so that the objective function is strictly increasing (in case of a maximization problem). If the length of the edge is finite, there is another basic feasible solution at the end. If the edge is infinite, the problem is unbounded and there is no optimal value to the objective function.

The nature of the simplex algorithm implies that an optimal solution (or the absence of it) can be reached within a finite number of computation steps.

The initial step (step 0) of the simplex method, is to transform the problem $\mathbf{Ax} \leq \mathbf{b}$ to a problem of the type $\mathbf{Ax} = \mathbf{b}$ by introducing slack variables. For example, if there is a constraint $x_1 + x_2 \leq 5$, it can be transformed to $x_1 + x_2 + x_3 = 5$ by means of the slack variable x_3 .

Following step 0, the first step of the simplex method is to find a starting point (the first basic feasible solution) for the algorithm. There are different methods depending on the problem to solve. For example, the Northwest Corner Rule that can be applied to find a basic feasible solution to the transportation problem among others (see pages 75 and 76 of [5]). In case, the original problem has only lower-than-or-equal signs in all constraints (as given in the standard formulation above), the slack variables give a first basic feasible solution. However, if this is



not the case, the process of finding a first basic feasible solution is not trivial and is normally the reason for not using the primal simplex algorithm (see section 3.3.2). If no basic feasible solution exists, the problem is called infeasible. Otherwise the algorithm enters the second phase, in which either an optimal solution to the problem is obtained, or the problem is found to be unbounded.

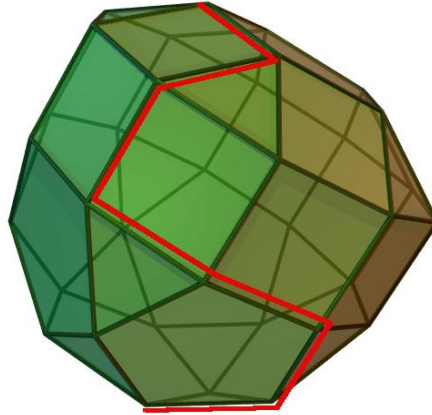


Figure 8: Graphic scheme of 3-dimensional simplex algorithm [6]

To understand the second simplex phase more in detail, it is necessary to first understand the concept of pivoting which is central to the simplex method. Assuming there is a basic feasible solution to the problem that is being solved, one can bring the linear system of equations $\mathbf{Ax} = \mathbf{b}$ into canonical form as shown in the following:

$$\begin{array}{cccccccccc}
 & \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \dots & \mathbf{x}_m & \mathbf{x}_{m+1} & \mathbf{x}_{m+2} & \dots & \mathbf{x}_n & \\
 \bar{\mathbf{a}}_1 & 1 & 0 & 0 & \dots & 0 & \bar{a}_{1(m+1)} & \bar{a}_{1(m+2)} & \dots & \bar{a}_{1n} & \bar{a}_{10} \\
 \bar{\mathbf{a}}_2 & 0 & 1 & 0 & \dots & 0 & \bar{a}_{2(m+1)} & \bar{a}_{2(m+2)} & \dots & \cdot & \bar{a}_{20} \\
 \cdot & 0 & 0 & 1 & \dots & \cdot & \cdot & \cdot & \dots & \cdot & \cdot \\
 \cdot & \cdot & \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \dots & \cdot & \cdot \\
 \cdot & \cdot & \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \dots & \cdot & \cdot \\
 \cdot & \cdot & \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \dots & \cdot & \cdot \\
 \bar{\mathbf{a}}_m & 0 & 0 & 0 & \dots & 1 & \bar{a}_{m(m+1)} & \bar{a}_{m(m+2)} & \dots & \bar{a}_{mn} & \bar{a}_{m0}
 \end{array} \tag{5}$$

The variables x_1 to x_m are the basic variables of the basic feasible solution (x_{m+1} to x_n are non-basic variables analogously) and \bar{a}_1 to \bar{a}_m are the rows of the modified matrix \bar{A} (without the last row). The last column of the tableau (\bar{a}_{10} to \bar{a}_{m0}) is equal to the basic entries of the solution vector \mathbf{x} . All remaining entries of \mathbf{x} are zero and do not appear in the tableau. The modified matrix \bar{A} is obtained from A by bringing the system $\mathbf{Ax} = \mathbf{b}$ into canonical form by means of Gaussian elimination operations.

To find another basic feasible solution, one of the basic variables, $x_p, 1 \leq p \leq m$, has to be replaced by a non-basic one, $x_q, (m+1) \leq q \leq n$. This is only possible if $\bar{a}_{pq} \neq 0$. To replace

the columns of the tableau and get a new basic feasible solution, the following formulas are applied. Denoting the coefficients of the new system in canonical form by \bar{a}'_{ij} , we get

$$\bar{a}'_{ij} = \bar{a}_{ij} - \frac{\bar{a}_{iq}}{\bar{a}_{pq}} * \bar{a}_{pj} , \quad i \neq p \quad (6)$$

and

$$\bar{a}'_{pj} = \frac{\bar{a}_{pj}}{\bar{a}_{pq}} \quad (7)$$

These equations are called Pivot Equations and are the tool of the simplex method to jump from one basic feasible solution to the next. However, the algorithm needs the additional information, on which element it needs to pivot, in order to find a new basic feasible solution with a better objective value. For this purpose, the tableau (5) is extended to the following form called the simplex tableau.

$$\begin{array}{ccccccccccc}
 & x_1 & x_2 & x_3 & \dots & x_m & x_{m+1} & x_{m+2} & \dots & x_n & \\
 \bar{a}_1 & 1 & 0 & 0 & \dots & 0 & \bar{a}_{1(m+1)} & \bar{a}_{1(m+2)} & \dots & \bar{a}_{1n} & \bar{a}_{10} \\
 \bar{a}_2 & 0 & 1 & 0 & \dots & 0 & \bar{a}_{2(m+1)} & \bar{a}_{2(m+2)} & \dots & \cdot & \bar{a}_{20} \\
 \cdot & 0 & 0 & 1 & \dots & \cdot & \cdot & \cdot & \dots & \cdot & \cdot \\
 \cdot & \cdot & \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \dots & \cdot & \cdot \\
 \cdot & \cdot & \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \dots & \cdot & \cdot \\
 \cdot & \cdot & \cdot & \cdot & \dots & \cdot & \cdot & \cdot & \dots & \cdot & \cdot \\
 \bar{a}_m & 0 & 0 & 0 & \dots & 1 & \bar{a}_{m(m+1)} & \bar{a}_{m(m+2)} & \dots & \bar{a}_{mn} & \bar{a}_{m0} \\
 r & 0 & 0 & 0 & \dots & 0 & r_{m+1} & r_{m+2} & \dots & r_n & -z_0
 \end{array} \quad (8)$$

The simplex tableau contains one additional row (here called r) with the relative cost coefficients r_j . This last row is the result of bringing the objective function into the form

$$c_1x_1 + c_2x_2 + \dots + c_nx_n - z = 0 \quad (9)$$

and appending it to tableau (5). Then the first m elements of the new row are eliminated by Gaussian elimination operations (in this case the objective value z is treated as another variable). The modified row looks like the following:

$$r_{m+1}x_{m+1} + r_{m+2}x_{m+2} + \dots + r_nx_n - z = -z_0 \quad (10)$$

Because z is treated like another variable and because it can be defined as a basic variable, the column corresponding to z does never change (0, 0, 0, ..., 1) and is therefore not included in the simplex tableau.

The relative cost coefficients r_j indicate whether there is a basic feasible solution with a better objective value or not. If, in a maximization problem, there is one relative cost coefficient that



is negative, then making the corresponding column a basic column will improve the objective. Analogously, in a minimization problem, a column with a positive relative cost coefficient is chosen. If there is more than one negative relative cost coefficient, either one of them can be used to choose a pivot element. Usually the most negative one is chosen. When all relative cost coefficients are non-negative, the current basic feasible solution is the optimal solution.

3.3.2. Dual simplex method

Corresponding to the primal problem formulation presented in the previous section, there is a dual formulation defined as

Minimize

$$\mathbf{b}^T * \mathbf{y}$$

Subject to

$$\mathbf{A}^T \mathbf{y} \geq \mathbf{c}, y_i \leq 0 \tag{11}$$

With

$$\begin{aligned} \mathbf{A} \in \mathbb{R}^{p \times n}, \quad \mathbf{y} = (x_1, \dots, x_n), \quad \mathbf{c} = (c_1, \dots, c_n) \text{ with } c_i \leq 0, \\ \mathbf{b} = (b_1, \dots, b_p) \end{aligned}$$

Where the roles of the objective function and the right-hand side of the constraints are reversed and the constraint matrix A is transposed. If the primal problem is a maximization problem, the dual is a minimization problem. Instead of the positive primal variable x , we use the negative dual variable y . This formulation can also be seen as the inverse of the primal. If the primal solution is unbounded, the dual problem is infeasible and vice versa. Both problem formulations have the same optimal value. Therefore, the feasible solutions of the dual formulation provide upper bounds on the primal simplex formulation.

As an example, in Figure 9 the primal and dual mechanisms are schematically displayed for a 2-dimensional case with the primal objective “maximize $x_1 + x_2$ ” and two linear constraints. Both problems tend towards the same optimal objective value from two sides. Therefore, optimizing the dual formulation is sometimes seen as a “backdoor” method that leads to the same optimal result.

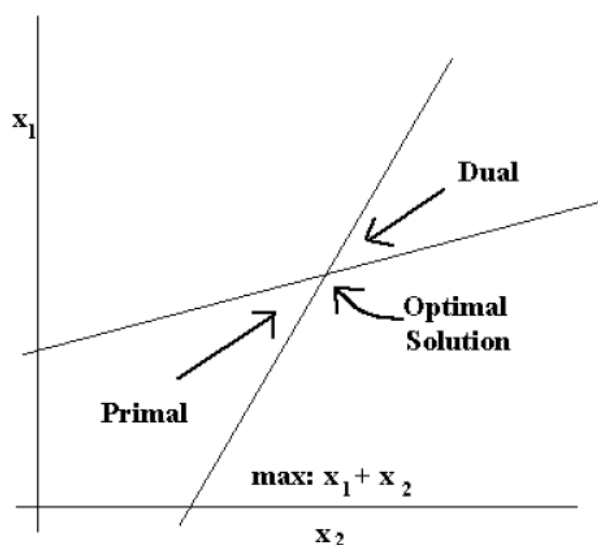


Figure 9: 2-D visualization of primal and dual simplex methods [7]

The dual simplex method optimizes the dual formulation of the optimization problem while actually using the primal simplex tableau. That has the advantage, that there is no need to reformulate the problem before optimizing it. However, using the dual simplex method requires two conditions to be true:

1. If the original (primal) problem formulation is a maximization problem, all the coefficients of the objective function need to be negative. Analogously, for a minimization problem, all the coefficients need to be positive. This is equivalent to saying that the problem prices out optimally.
2. At least one of the constraints needs to have a greater-than-or-equal or equal sign, which means that the first basic solution is infeasible.

In the next step, a pivot row is chosen from the simplex tableau (8) in the last section by looking at the right-hand side values of the constraint matrix a_{m0} . If there is a negative value, the corresponding row can be chosen to be the pivot row. The pivot element on this row can then be found by conducting the minimum test: Dividing all non-zero values of the last row "r" (objective function z) that has a corresponding negative value in the pivot row (same column), by this value without considering the sign. The element in the pivot row that yields the lowest result is the pivot element. The next step is, to make the pivot column a new basis vector by means of Gaussian eliminations. This way, by looking at the values a_{m0} instead of the relative cost coefficients r_i we solve the dual problem without actually formulating it.

As mentioned above, the dual simplex method works very well when there is a basis to the primal problem that "prices out" optimally (all the relative cost coefficients are non-negative for



a maximization problem, i.e. all the objective function coefficients are negative) but which yields an infeasible solution (one or more x_i are negative). Then this basis can be used for the dual simplex method to work towards feasibility while maintaining optimality, instead of spending more effort to look for a feasible basis and work towards optimality by maintaining feasibility with the primal simplex method. That is the reason why in many cases the dual simplex method is faster than the primal simplex method: less effort is necessary to find a suitable basis. The primal simplex method is usually faster, when there is a basic feasible solution to the problem available from the start. Otherwise, the effort of finding one, makes the dual simplex method more attractive.

More details about solving algorithms of linear problems can be found in [5].

3.4. Deployed modelling tools

Over the course of this thesis, different modelling tools were used to examine the behavior of the EMS under different conditions and to develop the EMS prototype developed at CITCEA towards a final product for GPM. Whereas Matlab and Simulink, as general engineering tools, are assumed to be known to the reader, the mathematical modeling tools are specific to optimization computing and might need additional explanation. Therefore, the modelling tools used in this thesis project are introduced in the following.

3.4.1. JuMP modeling language for mathematical optimization

JuMP is a mathematical modeling language embedded in the relatively new technical programming language Julia. It was released in 2012 and has still not reached version number 1. The language's performance can be compared to C or FORTRAN whereas it is as easy to use as for example Matlab.

JuMP is built on the Julia-concept of syntactic macros. This approach allows JuMP to avoid operator overloading. A concept that most of the other open source modeling languages use, which can, however, make the code very slow. An example is given in [8]:

Given is the expression $1 + \sum_{i=1}^d \sum_{j=1}^d |c_j - i|(1 - x_{i,j})x_{1,j}$ typed in directly in a high-level programming language. In the case of Python the expression takes advantage of the built in `sum()` command. In this case however, the partial sums are quadratic expressions with d^2 terms. Thus, the operation would result in d^2 summations because every term is accumulated one by one. In Julia, the expression could be written as a macro

```
@expr(1 + sum{abs(c[j]-i)*(1-x[i,j])*x[1,j], i in 1:N, j in 1:N})
```

('@' denotes a macro in Julia) keeping the easy to use syntax, which than internally runs a

routine that sums up all the partial sums at once, improving the performance significantly.

Therefore, in JuMP variable definitions or writing constraints is realized by calling macros (`@variable()`, `@constraint()`) allowing the user to type in the expressions very naturally and at the same time keeping a high model building performance.

Within the computer science community, macros have been recognized as a useful tool for developing domain-specific languages, of which JuMP is an example. [8]

3.4.2. CPLEX Concert technology – CPLEX's C++ API

In the last phase of this thesis project, it was decided to implement the EMS in C++, as this is the standard at GPM. A prerequisite for this decision, was the foregoing decision to use the CPLEX solver, as CPLEX was found to be the only possible solver for the OP of the EMS (see chapter 4). Therefore, the solver specific API of CPLEX could be deployed. It is worth noting at this point, that a change in the solver decision will result in the necessity for re-coding the EMS in the new solver specific API. Furthermore, after careful considerations, it was decided to develop the EMS in C++ for Windows platforms. The reason for that is, that the only devices that are powerful enough to solve the EMS in its current formulation, in a reasonable time frame, are GPM's micro servers that run on Windows operating systems. For this purpose, IBM provides a C++ API for the CPLEX optimizer, based on ILOG Concert Technology. This API contains all the functionality of CPLEX organized in classes and methods, also called objects. In the documentation [9], the structure of the API is described as in the following:

[Figure 10] shows a program using CPLEX Concert Technology to solve optimization problems. The optimization part of the user's application program is captured in a set of interacting C++ objects that the application creates and controls. These objects can be divided into two categories:

- *Modeling objects are used to define the optimization problem. Generally an application creates several modeling objects to specify the optimization problems. Those objects are grouped into an `IloModel` object representing the complete optimization problem.*
- *Solving objects in an instance of `IloCplex` are used to solve models created with the modeling objects. An instance of `IloCplex` reads a model and extracts its data to the appropriate representation for the CPLEX optimizers. Then the `IloCplex` object is ready to solve the model it extracted. After it solves a model, it can be queried for solution information.*



The CPLEX C++ API makes use of the pointer concept in C++. All instances of Ilo-classes in Concert Technology are only pointers referring to the actual memory location where the real object (created in the background) is saved. This implementation has the advantage, that the user can create several pointers, referring to the same memory location. However, the user must be careful to avoid creating empty pointers and to always delete objects, that are not needed anymore to avoid memory leaks.

More information on the decision for Visual C++ for Windows systems and the Concert API can be found in chapter 7 of this thesis.

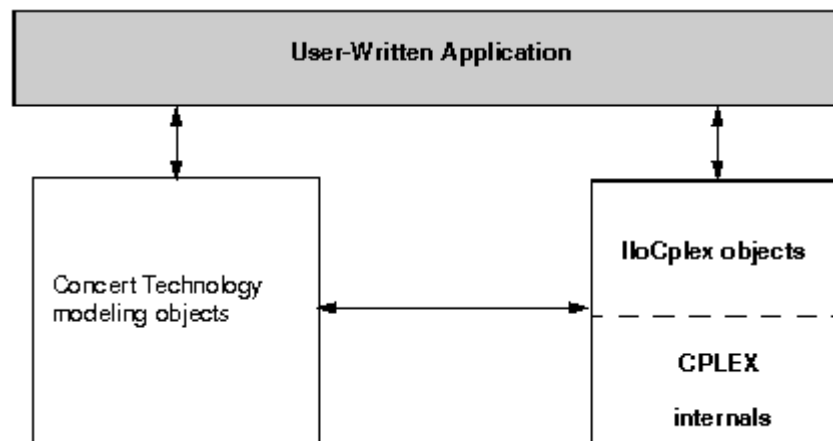


Figure 10: A view of Concert Technology for C++ users [9]

4. Decision on solver

4.1. Solver Tests

In order to decide whether an open source solver is sufficient to solve the EMS optimization problem or if it is too extensive and a commercial solver would be necessary, several quick tests were run to check the solver performance on this specific problem. The fact that the EMS optimization problem is modeled as a mixed-integer linear program (MILP) already gives a limited number of specialized solvers. For simplicity reasons, these first tests were run in GAMS where the problem was already formulated.

The tests on GAMS were run at the CITCEA lab for license reasons. The following five MILP solver licenses were available in the GAMS version on the used computer:

- **CPLEX**: Is a powerful commercial solver that can solve very large mixed-integer linear programs among others. It was first commercially available in 1988 from CPLEX Optimization Inc. which now belongs to IBM.
- **CBC**: **C**oin-or **B**ranch and **C**ut is an open source solver for MILPs available on the Coin-Or platform. It is considered to be one of the open-source solvers with the highest performance.
- **BDMLP**: Is a free LP and MILP solver available in the standard GAMS suit. It was originally developed by the World Bank but is now maintained by GAMS Corporation. The solver is not very powerful and is only intended for small to medium sized problems.
- **GLPK**: **G**NU **L**inear **P**rogramming **K**it is another free solver collection for large scale LPs and MILPs. It is maintained by the GNU project.
- **SCIP**: **S**olving **C**onstraint **I**nteger **P**rograms is the most powerful non-commercial solver currently available. In contrast to commercial solvers it gives the user wide low-level control over the solving process. SCIP is only free for academic use.

All the solvers deploy a branch- and cut algorithm based on one or several simplex methods for the LP relaxations (see sections 3.2 and 3.3).

The computer at CITCEA has the following data:

- Intel® Core™ i5-3330 CPU@ 3.00GHZ, 4 main cores



- 16GB RAM
- Microsoft Windows 7 Professional 64bit

The tests were run by solving the problem with three different versions of input data. Two of the input files (2 and 3) are from the beginning of a simulated day which is also the point with the largest optimization problem because the number of variables and constraints depends on the number of optimization executions left for the day (see section 3.1.4). The first input file is from the middle of the day (184/287 five-minute periods left). In the following diagram the results of the time to solve is plotted over the time-to-solve of CPLEX. Only the smaller input file was used to test all five solvers. On the second and third file only CPLEX and the fastest alternative solver (SCIP) were tested.

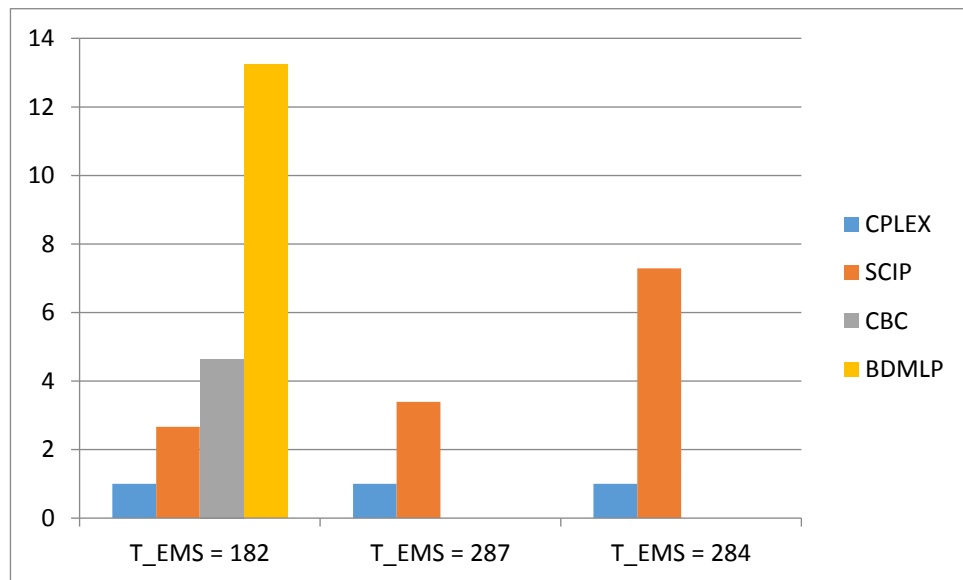


Figure 11: Results of solver test relative to CPLEX time

It can be seen that CPLEX is the fastest solver followed by SCIP, then CBC. The difference is significant. BDMLP is far behind the other solvers and GLPK did not manage to solve the model at all. The harder the problem gets the more difficulties the alternative solvers have in comparison to CPLEX. However, in test 2 and 3 SCIP solved the problem almost equally fast for both test whereas it CPLEX took significantly longer to solve test 2 in comparison to solve test 3.

The results acknowledge MILP solver test results from other sources. For example, in [10] test results from [11] are summarized. The test was run on real world test cases from the Netlib repository [12].

	running time	instances solved	solved (%)
CBC	10.20	41	47.13
CPLEX	1.45	73	83.91
GLPK	22.11	3	3.45
GUROBI	1.00	77	88.51
LP_SOLVE	19.40	5	5.75
SCIP-C	3.76	63	72.41
SCIP-L	6.40	52	59.77
SCIP-S	5.33	57	65.52
XPRESS	1.29	74	85.06

Figure 12: MILP solver test results from [11]

Also in this test, CPLEX was by far the best option from the ones tested for this thesis, in terms of running time as well as in terms of solved instances. The next best open source option is SCIP with SoPlex as underlying LP-solver (SCIP-S). It is still double as fast as CBC and solved 18% more test instances. GLPK only solved 3.45% of the test instances. This bad performance was also experienced during the GAMS test for this thesis. Among the commercial solvers in this test, GUROBI and XPRESS seem to perform slightly better than CPLEX. Here the deciding factor is the price for the license.

Based on the experiences during the GAMS tests and the presented extern test results, it was decided to use a commercial solver for this project.

To get a more reliable and detailed picture about the solver performance, more tests with CPLEX (and CBC) are performed in chapter 6. This quick test serves only to get a first picture of solver performances for the EMS optimization problem.

4.2. License considerations

As shown in the previous part, the optimal solution of the EMS OP (project case Cobija) requires the performance of a commercial solver due to its complexity and limited time frame. The leading commercial MILP solvers are CPLEX and Gurobi, but also XPRESS and MOSEK are popular commercial solvers in the market. These four solvers perform very similarly in terms of solving times and number of problems solved in many tests throughout the internet. As an example of such a test row, Figure 12 is given in the previous section. Considering the similar test results, the main decision element are license costs and availability. During the



course of this thesis, only academic licenses were an option and the only available licenses were the following:

- IBM ILOG CPLEX Optimization Studio 12.6.3 – Student
 - Windows X86-64
 - Linux X86-64
 - OSX
- IBM ILOG CPLEX Optimization Studio 12.7.1 – Student
 - Windows X86-64
 - Linux X86-64
 - Linux On System I/P
 - Linux On System Z
 - AIX
 - OSX

The former (older CPLEX) version is compatible with the currently available “CPLEX.jl” interface package for Julia/JuMP and was therefore used for the tests in chapter 6 of this thesis. The latter, which is the most recent version of CPLEX, is compatible with Microsoft Visual Studio 2017 installed on the student’s machine and therefore deployed in chapter 7 of this thesis.

5. EMS implementation in JuMP

In the original prototype developed at CITCEA, the optimization problem was solved in GAMS, which is a high-level modelling tool for mathematical optimization. It allows the user to formulate the problem in a language, which is very close to the native mathematical formulation. However, GAMS is a commercial software and therefore, apart from requiring a license for the solver (CPLEX in the case of this thesis), a license for GAMS itself is needed. Since this license is not available at GPM, an alternative to model the EMS optimization problem was required. There are several alternatives to the GAMS software.

1. Using another commercial modeling environment. The most famous representative is AMPL. However, here again it would be necessary to pursue a license.
2. Using an open source modeling environment.
3. Using the direct solver interfaces to C++. Since GPM products are written in that language, it would be the most direct way to call the solver. However, solver libraries differ from solver to solver. Therefore, the solver must be chosen first before implementing the optimization problem.

In the beginning of this thesis project, it was not clear which solver would be used in the final EMS implementation (see chapter 4). Therefore, at this point, option number 3 was not a possibility yet (in chapter 7, the EMS is implemented also with option number 3, using the CPLEX C++ interface). However, option 1 would not be optimal either, because the reason for not using GAMS anymore were the high license expenses. In an end product, the solver will be called directly, as proposed in option 3 (see chapter 7) and therefore, the modeling environment is only needed during the development phase of the EMS, while the Simulink simulation environment is used (see section 3.1.5). This means, that purchasing a license only for the development phase is not advisable, when other alternatives are available. Therefore, an open source modeling environment was chosen.

In [8] a benchmark analysis on the performance of different modeling environments was conducted. The studied models were quadratic and conic quadratic respectively of different sizes. In Figure 13, the elapsed times (in seconds) between launching the executable that builds the model and the start of the solver are compared. Since Gurobi 6.0.0 was chosen as a solver in the source, also the direct interface of Gurobi and C++ (GRB/C++) was included in the test.



Instance	JuMP	Commercial			Open-source		
		GRB/C++	AMPL	GAMS	Pyomo	CVX	YALMIP
lqcp-500	8	2	2	3	53	8	19
lqcp-1000	10	6	5	13	214	43	115
lqcp-1500	13	14	12	42	500	120	400
lqcp-2000	18	26	22	102	>600	267	>600
fac-25	7	0	0	0	15	>600	465
fac-50	9	2	2	4	110	>600	>600
fac-75	13	6	7	11	376	>600	>600
fac-100	23	12	17	25	>600	>600	>600

Figure 13: Benchmark analysis results of different modeling environments in seconds to build the model [8]

The results show that JuMP has a performance that is comparable to commercial modeling languages, especially for very large models. The EMS optimization model can be considered as very large. All the other open source languages tested in the paper turned out to be considerably inferior. The reason for this is JuMP's syntactic macro approach based on Julia technology in contrast to the alternative's operator overloading approach (see section 3.4.1).

Another important factor when choosing a modeling language is its suitability to write programs. The modeling language JuMP is embedded in Julia which is a very recent language for technical computing. When developing Julia, the focus was set to provide performance comparable to low-level languages while at the same time be as easy to use as for example Matlab. JuMP benefits significantly from these properties of Julia. Because commercial modeling environments are designed as stand-alone applications and because their syntax and data structures are very specialized for modeling mathematical optimization problems, they have shortcomings when it comes to embedding the optimization into a larger project. Loading and saving external data to and from other programs is difficult.

In case of the EMS, the conversion to and from the GDX-files for the GAMS optimization became completely obsolete after the implementation in JuMP, because it accepts inputs in the Matlab format by means of the MAT package in Julia. This leads to another advantage of open source software: Packages are constantly added and the functionality expands quickly. By using JuMP, it is possible to reduce the size and complexity of the EMS optimization model, but even more importantly, to omit the interface between GAMS and Matlab. At the same time, the solver selection process is kept simple. To choose another solver, only one line of code needs to be adapted. That makes it easy to compare different solvers during the course of this thesis and beyond. In conclusion, it can be noted, that implementing the EMS in JuMP instead of GAMS has many advantages without any drawbacks.

During the development phase, the JuMP implementation is also superior to the C++ version

developed in chapter 7. As described above, it is very intuitively linkable to Matlab, in which the simulation of the power plant is run. Furthermore, JuMP takes advantage of the Julia workspace, in which all variables used by the program are saved and hence, visible for the user. The Julia workspace is comparable to the Matlab workspace. This provides the user with a powerful tool, to explore different configurations of the EMS OP and to find bugs inserted during the development phase. Concluding, it is advisable to use the JuMP implementation while working with simulations, to find the best EMS layout for a specific plant. Only afterwards, the found layout should be included in the C++ implementation.

5.1. Implementation of the mathematical model

To avoid the need for a GAMS license during the development phase of the final EMS, the problem formulation was ported from GAMS to JuMP which is a modelling toolkit in the Julia language (see section 3.4.1). The solver was decided to be CPLEX (see chapter 4). Under these conditions three Julia packages are needed to perform the solution of the optimization problem:

- JuMP (**J**ulia for **M**athematical **O**ptimization) brings language features to model optimization problems based on the Julia concept of macros. It currently supports a number of solvers for different classes of optimization problems
- CPLEX provides an unofficial interface to the popular solver by IBM. There needs to be a working installation of the CPLEX optimizer installed on the machine and a valid license. Solver options can be handed over in the CPLEX C-syntax.
- MAT makes it possible to read and write mat-files in Julia. This makes it very easy to link the Julia program to the main part in Matlab/Simulink.

The packages can be installed by using the Julia package manager. The syntax is (exemplary for the JuMP package):

```
Pkg.add("JuMP")
```

Then on top of the script the packages are included by writing:

```
using JuMP, MAT, CPLEX
```

The first step in the jl.-file execution is to import the file "OP_input.mat". This file is created in every period of the EMS and contains all the sets and parameters necessary to build the model that has to be solved by CPLEX. The file is loaded with the command




```
data = matread("OP_input.mat")
```

which stores the data into a Dict-object called “data”. This kind of object is a look-up table where every parameter or array of parameters is mapped to a name. The matread-command stores the Matlab vectors and matrices as Julia arrays into the Dict with their key being their Matlab identifier. Therefore, in the next step, these arrays and values can be extracted as exemplarily shown in the following for the minimum frequency allowed in the system:

```
f_mn = data["par_f_mn"]
```

Now a JuMP optimization model object can be created and a solver can be assigned to the model. In this case, the CPLEX optimizer is used through the Julia CPLEX package and a solver time limit is set. Over the course of the examinations, it has been found effective to set the time limit 20 seconds before the next optimization period starts, in order to completely finish the previous optimization before the start of the next one. Experiments with and without a solver time limit were conducted and no significant difference was found in the results (see section 6.2.1). In the following the syntax for the model assignment, solver choice, and solver time limit is shown:

```
m = Model(solver = CplexSolver(CPX_PARAM_TILIM = T_intra*30-20))
```

As can be easily seen in the code snippet above, to change the solver only requires the user to change a single line of code. That makes JuMP (as a free alternative to GAMS or AMPL) very interesting for this kind of research when the final decision about the solver choice is not yet reached.

Remark: In section 6.4, it was made use of this functionality of JuMP: the open source solver CBC could be tested again under new conditions by only adapting the code line shown above.

In the next steps, variables, constraints, and the objective function are added to the model called ‘m’. In JuMP, this is done by means of Julia macros denoted by the operator ‘@’. In the case of the EMS project, the mathematical model of the optimization problem has variable- and constraint arrays (see section 3.1.4). These arrays can be naturally created in JuMP which significantly simplifies the model creation. In the following, it is exemplarily shown for the variable array ‘P_bat’ how to create the array and how to create a constraint array linking the variable arrays ‘P_bat’, ‘P_bat_car’, and ‘P_bat_desc’ in order to model the different algebraic sign of the battery power when charging or discharging the battery.

```
@variable(m, P_mnB <= P_bat[1:T_EMS,1:T_intra] <= P_mxB )
```

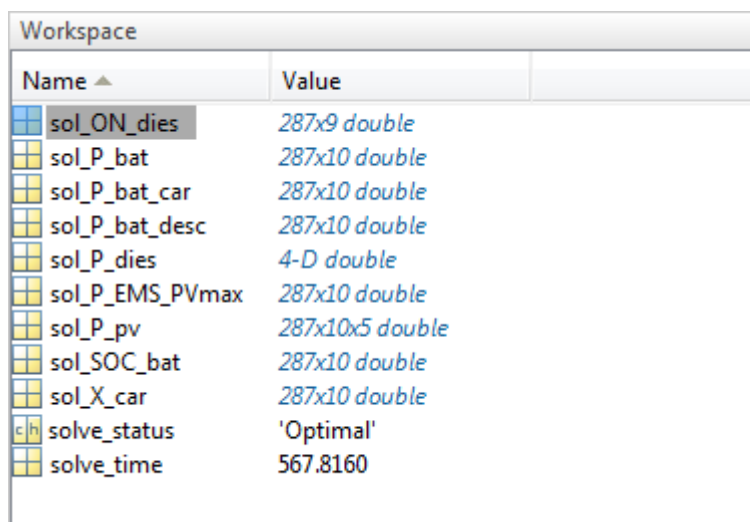
```
@constraint(m, Bat3[t=1:T_EMS,p=1:T_intra], P_bat[t,p] == P_bat_car[t,p] - P_bat_desc[t,p])
```

The variable bounds are set in the variable definition between the input parameters of the optimization problem 'P_mnb' and 'P_mxb'. The decision variable 'P_bat' as well as the constraint array 'Bat3' are two-dimensional with T_EMS (number of remaining optimization problem executions) times T_intra (number of setpoint dispatches between two executions) entries. The objective function is created in the same way using the macro '@objective(...)'.

After having defined the objective for the model 'm', the model is complete. By executing the command

```
status = solve(m)
```

The model is handed over to the solver together with the solver options and the solver output is printed in the console. The solution of the model is the main part of the program in respect to execution time. After CPLEX finishes execution it returns a status (for example 'Optimal' if the solver was able to solve to optimality). As last step of the Julia script, the decision variables calculated by the solver, and the parameters 'solve_time' (elapsed time in seconds of the sum of all operations executed by the CPLEX optimizer to solve the model), and 'solve_status' (quality of solution reported by the CPLEX optimizer. E.g. 'Optimal' or 'AbortTimeLim') are printed into the file 'OP_results.mat' by means of the MAT-package functions 'matopen()', 'write()', and 'close()'.



Name	Value
sol_ON_dies	287x9 double
sol_P_bat	287x10 double
sol_P_bat_car	287x10 double
sol_P_bat_desc	287x10 double
sol_P_dies	4-D double
sol_P_EMS_PVmax	287x10 double
sol_P_pv	287x10x5 double
sol_SOC_bat	287x10 double
sol_X_car	287x10 double
solve_status	'Optimal'
solve_time	567.8160

Figure 14: optimization output file 'OP_results.mat' loaded into the Matlab workspace

An overview over the program flow of 'Optimization.jl' is given below.



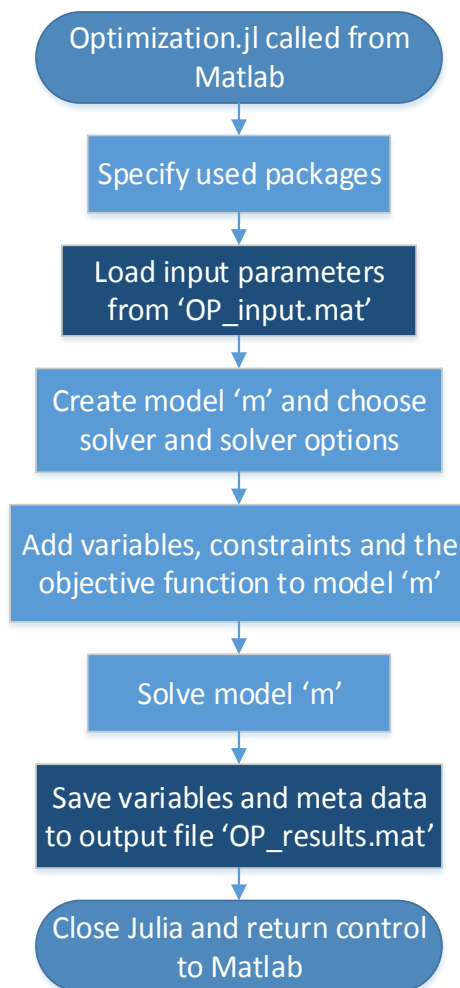


Figure 15: Schematic program flow of the of the OP execution script 'Optimization.jl'

5.2. Embedding the Julia script

As in the original implementation by CITCEA, the optimization script is embedded in a Matlab script that prepares the input file, executes the OP script, and reads the output file with the decision variables/setpoints. Depending on whether the optimization is executed during a simulation or during an emulation with real power converters, the embedding script differs. In the following a description of the different implementations is given.

5.2.1. Embedding the OP for simulations

The Simulink simulation model in which the EMS was developed and all the corresponding Matlab functions were developed at the CITCEA laboratories. However, the original implementation called GAMS to solve the optimization problem (see introduction of this chapter). Therefore, the embedding Matlab function needed to be adapted. The schematic

structure of how the OP execution is embedded in the simulation flow is displayed in Figure 16. In this example, the OP period is the original one with 5 minutes between the OP executions. That means that within one day, the OP is solved 288 times. The sequence of actions is elaborated in the following:



Figure 16: Scheme of Simulink simulation

In a first step, the parameters defined by the user are loaded from a csv-file in the same folder as the M-function. These parameters are

- The scaling factor to scale the PV generation up or down in order to match the consumption and to create more interesting simulations to test the EMS
- The OP execution period “op_exe_step” that is used to calculate indices for the optimization
- The resolution of the forecast data “forc_res_step”. During this thesis project, it is always 35 minutes.
- The mode selection variable “acc_sub_opt”. If it is set to 1, the program will accept CPLEX results that were interrupted by a CPLEX time limit and not solved to optimality. If it is set to 0, the simulation will be continued with setpoints calculated in a previous OP execution.

After defining the indices for the OP execution and forecast data by means of “op_exe_step”, “forc_res_step”, and the simulation time which is an input parameter to the M-function, the forecast data is loaded into the workspace. This forecast needs to be transformed into several scenarios, for the solver to take into account the uncertainty of the decision (see section 3.1.2). For the PV generation, a matrix with five forecast scenarios is generated by the R-script “Generacio Escenaris corregit.R” (see section 3.1.5). The only modification made during simulation runtime is to replace small negative values by 0+ (very small positive values) to make the OP feasible at all times. This step is necessary because one of the constraints of the OP forces the PV power to take values between 0W and the maximum forecasted/allowed values. The scenarios for the consumption however are generated during simulation runtime. For that purpose, the loaded forecast data is modified by means of the Matlab function “normrnd(μ, σ)”. The variance σ^2 of the consumption data is calculated as described in sections



3.1.2 and 3.1.5. μ is 0. This function generates normal distributed random numbers that are added to the consumption forecast data as many times as there are scenarios. Dimensions of the two forecast containers handed over to the Julia script are

N° remaining OP executions $\times N^{\circ}$ 30s periods within OP exe step $\times N^{\circ}$ scenarios

As an example, with OP execution step = 5, number of scenarios = 5, and OP index 1, we obtain 287x10x5.

In addition to the scenario containers, more input parameters are defined in the M-function:

- Number of scenarios
- Number of remaining OP executions for the day
- Number of 30s periods within two OP executions
- Number of diesel generators in the system
- Min/max values of SOC, diesel power, and battery power
- Battery capacity in kW*30sec
- Minimum allowed system frequency
- Actual SOC of battery handed over from Simulink as second input parameter to the M-function

In total, 15 parameters are saved to the mat-file OP_input.mat.

In the next step, the Julia script is executed as a shell escape command. The script is run and Matlab waits for its completion to continue executing the M-function. After the optimization is finished and the results are saved in OP_results.mat, the script continues by loading all the setpoint arrays into the workspace. In the last step, these arrays are extracted and the necessary setpoints (number of running diesel generators, maximum PV output power, and battery power) are handed back over to Simulink as output parameters, whereas only the setpoints needed until the next OP execution are extracted. For instance, if the OP execution period is 5 minutes long, only 10 setpoints per control are handed over.

In case, the user sets "acc_sub_opt" to 0 and the most recent OP results are not optimal, the last result file with optimal results is loaded and the corresponding setpoints are extracted from there. The optimality check is performed by reading the variable "solve_status" in the result file. CPLEX replies "OPTIMAL" if the OP was completely solved, and "ABORT_TIME_LIM" if the solving process was interrupted by a time limit set by the user. In case the new OP results are used for the simulation, the results are saved in the result folder, from where they can be loaded, if the future results are not optimal.

The scheme of the M-function described in this part is also shown in Figure 17.

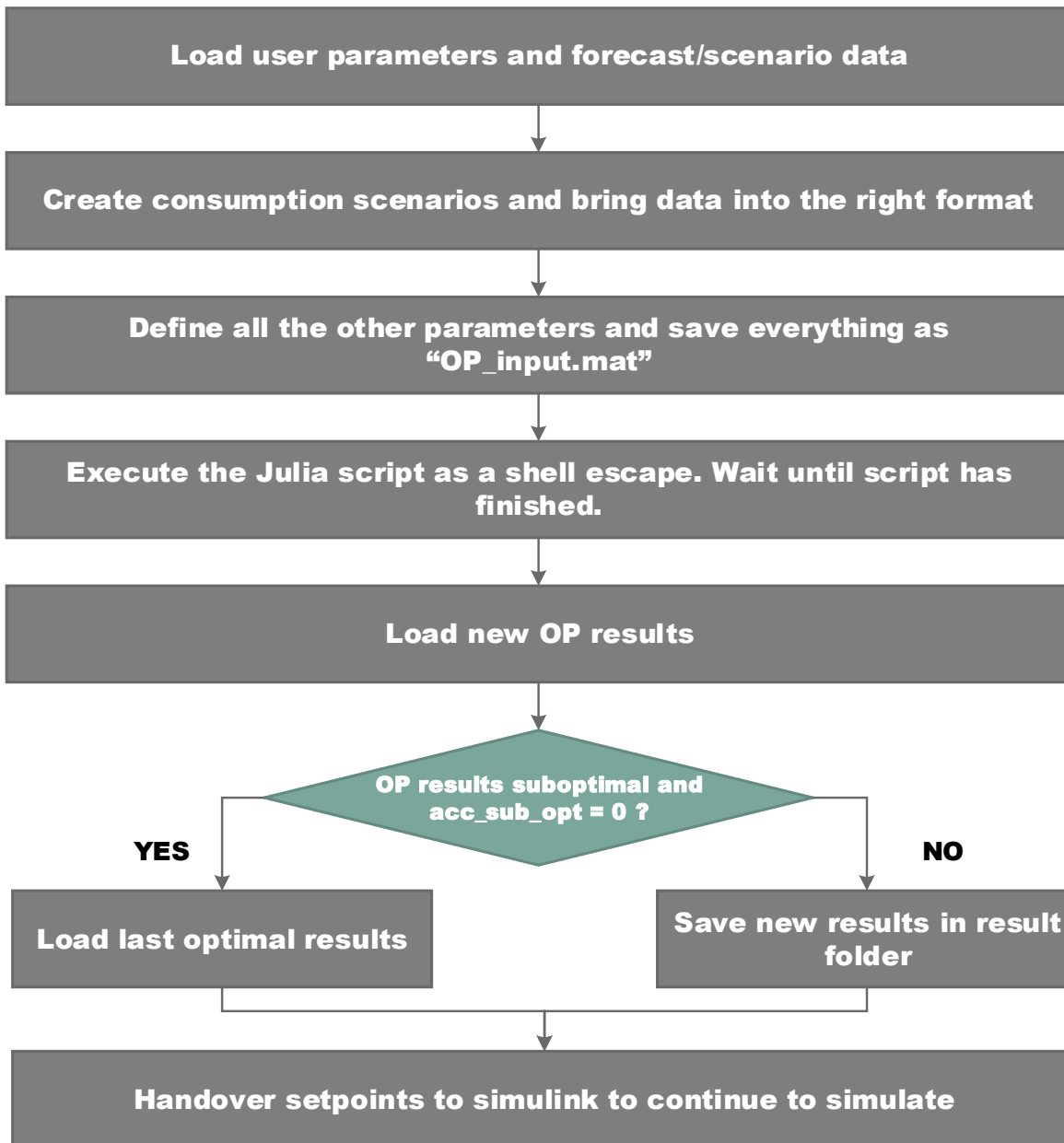


Figure 17: Flow chart of M-function `opti_handover.m`

5.2.2. Embedding the OP for emulations

The main difference when comparing the simulation on a computer with a real-world test or an emulation with real power converters is, that the simulation is paused during the OP execution in a Simulink simulation. In a real-world case, the problem must be solved before the start of the next OP execution to generate setpoints for the next OP execution period. That means that the current SOC is used to calculate setpoints during the next OP period to use in the period



after the next, instead of stopping the simulation and solving the optimization to use during the next OP period. That means, it has to be made sure that the optimization finishes in time to be able to send new setpoints to the converters. This can be assured by setting a time limit for the solver. Another possibility is, to send setpoints from the last OP execution that was successful, similar to the implementation in the previous chapter (see Figure 17). The difference here is, that the setpoints are sent individually every 30 seconds instead of all together every OP execution period. Therefore, every 30 seconds, the script first checks, whether the previous OP execution had finished in time, and if not, if it has now. If yes, the script continues with the newest setpoints.

Only after this part, a new OP execution is initialized, if a new OP execution period starts. That also includes the loading of scenarios and initialization of OP input parameters as described in section 5.2.1. If, for instance, the OP execution period is 5 minutes, the script is executed 9 times during that period without initializing a new OP execution, just to send new setpoints to the converters, and to check whether to use the old results, as long as a delayed optimization run has not finished yet. Only every tenth execution executes the Julia script. The same holds true for the dispatch of the setpoint for the number of diesel generators that are turned on.

The dispatch of the setpoints is realized by means of the `tcpip`-function in Matlab, that creates a `tcpip`-object, that can be written on and queried for data. The format of this data sent over the `tcpip`-object is the Modbus protocol. For this purpose, external functions are used, to convert the Matlab data into Modbus compatible data and reverse. These functions need the `tcpip`-object, the Modbus address, and the data to send (in case of sending data), as input parameters.

The execution of this Matlab script is managed by a Simulink timer object that executes the script every 30 seconds.

The basis of this script was developed at CITCEA to run emulations on the basis of Matlab and GAMS. In the course of this thesis project, the script was changed to work with the Julia script developed in this project.

6. Examinations for the final EMS implementation

One of the objectives of this thesis is to examine the computing resources that are needed for the EMS to work satisfactory. Since the required time to solve the OP primarily depends on the deployed hardware, there is not a final decision on the exact design of the EMS until the hardware platform is chosen. On the other hand, the layout of the EMS and its performance can help to decide on the hardware platform needed (see section 7). At this stage of the project, it is not constructive to take a final decision about the hardware or the exact layout of the EMS. Instead, this chapter examines the advantages and disadvantages of different EMS layouts and compares them. This way, in a later implementation phase, one or several of the presented layouts can be applied/combined to make the EMS work properly in the selected environment.

The computer used for the testing of different EMS layouts has the following properties:

- Microsoft Windows 7 Professional
- X64 based system
- Intel® Core™ i3-2100 CPU @ 3.10GHz, 4 logical cores
- 4GB RAM

On this platform, even though the CPLEX optimizer is one of the fastest optimizers available, it is not able to solve the basic layout of the OP for the test case “Cobija” to optimality in under five minutes in the beginning of the day, when the optimization needs to take into account most of the day to come. This means, that the optimal set points for the converters are not computed in time.

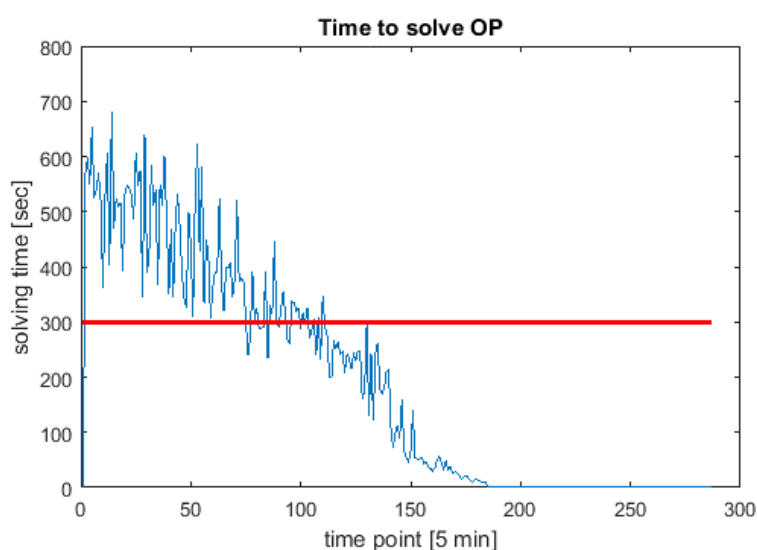


Figure 18: Time to solve all OPs over one day - Dataset "Data 1"



In the following, different approaches to reduce OP solve times or increase the OP time frame and their effects on the results are presented. The criteria to compare different simulations is the total PV power generated over the day, and the battery losses that are proportional to the energy that flows into or out of the battery. Therefore, to evaluate the results and compare different settings of the EMS, the result of the formula

$$E_s = \sum_{t=1}^T (P_{PV} - (1 - \eta_{bat}) P_{bat}) t \quad (12)$$

which is the amount of energy saved compared to the scenario with only diesel generators. It is essentially the objective function of the optimization problem that is solved by CPLEX repeatedly, with the difference that only the result is evaluated and therefore, no scenarios appear in the formula. EMS settings that result in higher E_s are generally considered better than settings with lower E_s (exceptional case number of scenarios in section 6.3).

For the tests, three different PV generation data sets were used:

1. “Data 1” is the basic data set that was used to develop the EMS. It was measured during a day with clear sky in the morning (low volatility) and some clouds in the afternoon (high volatility). The data is scaled up with a scaling factor of 2 in order to match the consumption data. Maximum value: 8134 kW.
2. “Data 3” was measured during a day with high solar power but also high volatility (partly cloudy summer day). It has the most significant generation drops of the three data sets. The data is scaled down with a scaling factor of 0.6 in order to match the consumption data. Maximum value: 9592 kW.
3. “Data 5” was measured during a completely sunny day. That means that there is no volatility and the curve is very smooth. The data is scaled up with a scaling factor of 1.2 in order to match the consumption data. Maximum value: 7590 kW.

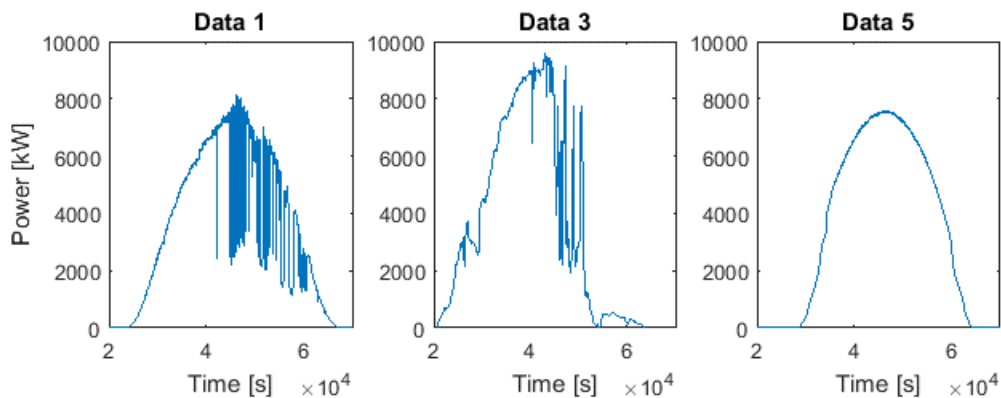


Figure 19: Scaled generation data sets used in this thesis

In order to determine the deviations that occur between several runs of the same simulations, first tests were run by just repeating simulations of the base case (OP execution period = 5 minutes, no time limit). Because neither the solver itself, nor the simulation work completely deterministic, the same simulations can yield different results. The difference between these results is then used to decide, whether the measures tested in the following sections improve the EMS yield considerably or not. The results of these preliminary test are displayed in Table 2. Additionally, the case without the EMS is shown in the table, when the battery does only help to stabilize the system frequency instead of receiving optimized setpoints from the EMS.

		Test 1	Test 2	Test 3	Max. Deviation	No EMS
E_s [MWh]	Data 1	44.232	44.053	44.152	0.179	37.710
	Data 3	43.341	43.659	43.509	0.318	30.737
	Data 5	47.530	47.244	47.003	0.527	36.263

Table 2: Deviations of solution for identic simulations

As can be seen, the deviations range between 0.5% and 1% of the total utility of the EMS. Another observation is, that the more predictable the PV generation is, the higher becomes E_s .

6.1. Adapting the OP execution period

The most obvious way to give the optimizer more time to reach an optimal solution is to increase the time interval for the optimizer. At the start of this thesis, the optimization period was a fixed period of five minutes. In order to change that interval, a variable called 'op_exe_step' was added to adjust the indices and time steps that are handed over to the optimizer. To give an example: if the optimization interval is increased to 10 minutes instead of 5 minutes, there will be 20 setpoint updates instead of only 10, because the setpoints are updated every 30 seconds. That means, the index parameter 'T_intra' that is handed over to the Optimization.jl script, changes from 10 to 20, whereas the maximum value that the parameter 'T_EMS' will take, is 144 instead of 288 (first OP execution of the day, 144/288 10/5-minute intervals remaining until next day).

To test the influence of this variable on the results, two additional simulations where run with the same simulation data but different OP periods. Additionally, the same simulations where run with two different PV generation data sets. That results in nine simulations in total. The three periods were 5 minutes, 10 minutes, and 15 minutes. The resulting E_s is shown in the following table.



	OP execution period	5 minutes	10 minutes	15 minutes
E_s [MWh]	Data 1	44.232	44.110	44.102
	Data 3	43.341	43.079	42.674
	Data 5	47.530	46.899	47.214
SUM		134.681	134.088	133.990

Table 3: Comparison of different OP execution periods

It can be seen, that the resulting E_s are relatively close to each other. Moreover, it is not always one period, that yields the best results, as in the dataset “Data 3” has the best value at an OP execution period of 10 minutes. However, the differences lie within the tolerances shown in Table 2. This means that there is no significant quality difference between the cases.

In Figure 20, solving times during the course of the 9 simulations of Table 3 are displayed. The arrangement of the plot corresponds to the order in Table 3, row 1 to 3 being the three data sets, and column 1 to 3 being the three OP execution periods. The red line marks the end of the OP execution period, when the solver would need to be finished in a real-world application.

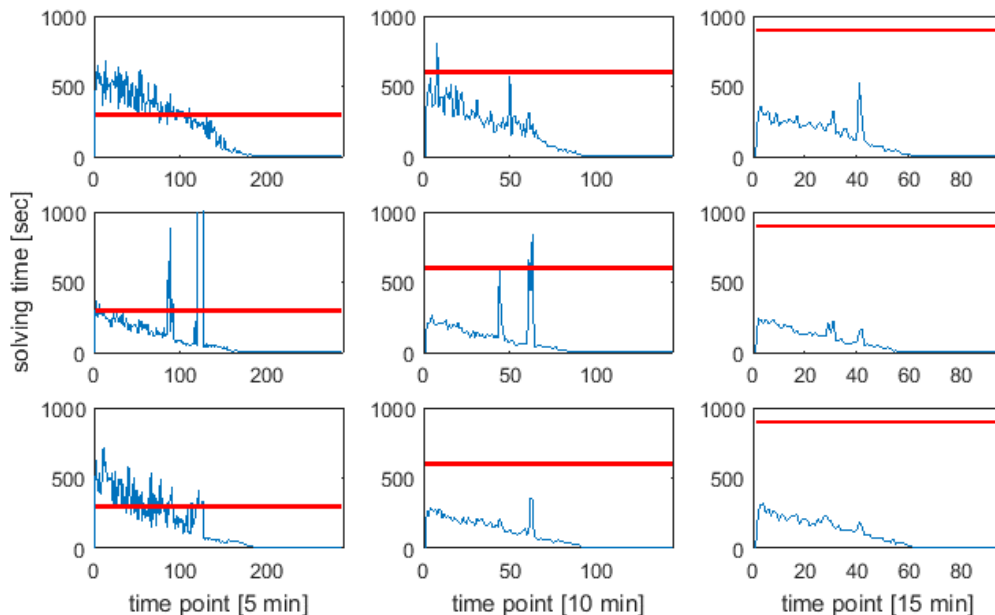


Figure 20: Comparison of OP solving times for different OP execution periods and different data sets

The first observation is, that increasing the OP execution period, generally reduces the complexity of the problem. That means, that the solver gets more time to solve the problem, and additionally, the problem can be solved in less time. These two effects add up. Furthermore, spikes in computation time are reduced when increasing the OP execution period.

Another observation is that the time needed by the solver, and the obtainable benefit by increasing the OP execution period, depend on the input data of the model. On a day with less volatility (Data 5/last row), the benefit of increasing the period is higher than on a day with high volatility (Data1/first row).

It should be recalled here, that the fact, whether the OP could be solved in time or not, is of limited interest, because it depends on the deployed hardware. The conclusions are of qualitative nature.

6.2. Enforcing the OP execution period

In the real case, in contrast to running simulations in Simulink, the OP time period is a hard deadline, after which new setpoints need to be available. In the following, two different possibilities to ensure this, are presented and compared. It has to be noted, that the results will differ more, if the machine that solves the optimization has less performance. If, on the other hand, the machine is powerful enough to solve the OP in time for every OP step during one day, the cases are identical to the base case.

6.2.1. Setting a time limit in the CPLEX optimizer

The CPLEX optimizer provides the option to limit the time the optimizer has, to compute a solution to the optimization problem. For this purpose, CPLEX provides the option parameter 'CPX_PARAM_TILIM'. The time limit needs to be inserted in seconds. In this implementation, the limit was set to 20 seconds before the start of the next OP period in order to give the system enough time to generate, save, read, and remove all the previous files. The implementation is shown in the following:

```
m = Model(solver = CplexSolver(CPX_PARAM_TILIM = T_intra*30-20))
```

If the time limit runs out before CPLEX has found an optimal solution, CPLEX will return the variables and objective value for the best feasible solution it has found up to that point. Furthermore, the status variable will be set to 'AbortTimeLim'.

The effect of the time limit is illustrated in Figure 21 exemplary for the data set "Data 1" and an OP execution period of 5 minutes. The results are shown in Table 4.



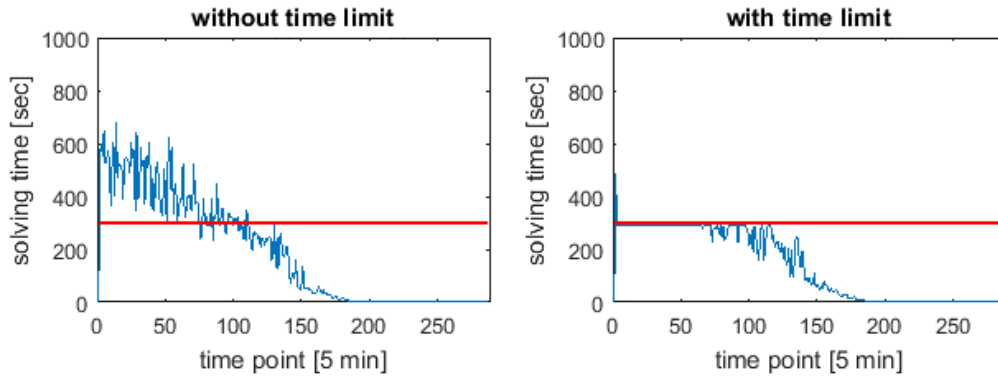


Figure 21: Comparison of OP solving times - with and without time limit

6.2.2. Using previous results

Since the optimization is solved every time considering the remaining day, all the setpoints for the remaining day from the perspective of the current OP period (actual forecasts, scenarios, SOC) are available in every generated result file. Therefore, there is the possibility to extract the new setpoints from an older file, in case the OP was not solved in time, and there does not exist a new result file. The results are compared with the suboptimal results from the aborted optimization, and the base case in the following table:

		No time limit	CPLEX time limit	Use previous results
E_s [MWh]	Data 1	44.232	44.369	44.315
	Data 3	43.341	43.004	43.548
	Data 5	47.530	47.558	47.751

Table 4: Comparison of base case (no time limit), time limited results, and using older results instead

6.2.3. Discussion and comparison

Comparing the results displayed in Table 4, it can be noted that neither the suboptimal solution, nor using the previous result, decreases the quality of the EMS in this test case. The reason for this is, that CPLEX already finds very good incumbents, way before the problem is solved to optimality. These incumbents have an offset of 1-2% to the root relaxation objective. In Figure 22, one can see the typical jump of the gap when CPLEX finds the first integral solution while exploring the search tree: In the 3rd column one can see the objective value of the current relaxation, in the 5th column the objective of the best incumbent, and in the 6th column the

objective of the dual problem (upper bound). The gap (in the last column) is the difference between the upper bound and the incumbent's objective in percent. From the moment of the jump from 18.76% to 0.08%, the solution can already be regarded as technically optimal for the EMS case. This is why the simulation results for the CPLEX time limit and the base case in Table 4 are technically identical.

The reason why the results for using the previous results are not worse either is very similar: between two OP execution periods, the circumstances do not change as much as to considerable yield different results for the optimization. This might change, if the forecast is updated more frequently.

Considering the low effort of setting a time limit for the solver and the low impact that this normally has on the results, it can be concluded, that setting a time limit is a very good option in many cases to keep the solution time of the OP under control.

```

285 259 2.59434e+007 58 2.18592e+007 2.59608e+007 257808 18.76%
Elapsed time = 265.84 sec. (112990.91 ticks, tree = 12.88 MB, solutions = 2)
320 294 2.59401e+007 56 2.18592e+007 2.59608e+007 261940 18.76%
323 297 2.59258e+007 48 2.18592e+007 2.59608e+007 262020 18.76%
324 298 2.59401e+007 51 2.18592e+007 2.59608e+007 262066 18.76%
325 299 2.59096e+007 35 2.18592e+007 2.59608e+007 262073 18.76%
326 300 2.59448e+007 44 2.18592e+007 2.59608e+007 262081 18.76%
327 301 2.59180e+007 55 2.18592e+007 2.59608e+007 262859 18.76%
328 302 2.59401e+007 50 2.18592e+007 2.59608e+007 262870 18.76%
329 303 2.59096e+007 31 2.18592e+007 2.59608e+007 262876 18.76%
330 304 2.59447e+007 43 2.18592e+007 2.59608e+007 262891 18.76%
426 395 2.59028e+007 33 2.18592e+007 2.59608e+007 271593 18.76%
Elapsed time = 277.34 sec. (116855.90 ticks, tree = 24.65 MB, solutions = 2)
* 502 458 integral 0 2.59407e+007 2.59608e+007 279383 0.08%
506 462 2.59509e+007 129 2.59407e+007 2.59608e+007 281497 0.08%
508 463 2.59355e+007 11 2.59407e+007 2.59608e+007 281618 0.08%
510 465 2.59508e+007 127 2.59407e+007 2.59608e+007 281664 0.08%
512 467 2.59333e+007 11 2.59407e+007 2.59608e+007 281742 0.08%
514 469 2.59508e+007 127 2.59407e+007 2.59608e+007 282106 0.08%
516 471 2.59355e+007 10 2.59407e+007 2.59608e+007 282144 0.08%
518 473 2.59507e+007 126 2.59407e+007 2.59608e+007 282245 0.08%
520 475 2.59355e+007 9 2.59407e+007 2.59608e+007 282964 0.08%
522 477 2.59507e+007 125 2.59407e+007 2.59608e+007 283347 0.08%
Elapsed time = 291.07 sec. (121513.87 ticks, tree = 28.78 MB, solutions = 2)

```

Figure 22: CPLEX output of first OP execution of the day

6.3. Reducing the number of scenarios

It is possible to reduce the complexity of the mathematical model of the EMS by considering less scenarios for consumption and generation. The size of 2 of 9 variable arrays (P_diesel and P_pv) is proportional to the number of scenarios. The same applies to 5 of 11 constraint arrays and for the objective function. This shows, that the impact of reducing the number of scenarios on the problem size is significant. However, the reduction of scenarios comes with the cost of reducing the variability that the EMS can consider. That means, the less scenarios are considered in the optimization, the more likely the frequency restrictions will be violated



when facing big power fluctuations. In other words: The less scenarios are considered, the larger the power reserves for the PPC need to be. This leads to the conclusion, that another optimization problem needs to be solved when setting up a concrete project containing the EMS optimizing:

- Number of scenarios
- Optimal power reserves
 - Minimum number of connected diesel
 - Maximum power setpoint for diesels
 - Battery SOC reserve
 - Battery power reserve

The objective of this optimization can be the same as in the EMS OP, measuring the PV energy used minus the battery losses. This optimization will require extensive historic data of the specific power plant. As a second step, on the basis of this optimization, hardware and solver for the EMS can be chosen accordingly.

The yield-increase for the EMS for this test case due to reducing the number of scenarios is illustrated exemplary for Data 3 in Table 5. In this test case, the scenario reduction did not significantly change the frequency behavior. This is due to relatively large power reserves chosen for the EMS. As can be observed, less scenarios increase the EMS yield, as long as the forecast is mostly correct.

Number of Scenarios	5	4	3
E_s [MWh]	42.919	43.170	43.330

Table 5: E_s in MWh for different numbers of considered scenarios for Data 3

As shown in Figure 23 (exemplary for Data 3), the reduction of the OP by only one scenario has a significant influence on the computation time for the EMS in the test case of this thesis project. The difference amounts to about 100 seconds computation time per scenario in the beginning of the day.

Concluding it can be said, that the number of scenarios is a variable, which needs to be optimally determined for every new EMS project, depending on the specific power plant and consumption/weather patterns, that apply. It has the potential to significantly change the complexity/computational costs of the OP. The algorithm to determine the optimal number of scenarios lies beyond the scope of this thesis and will need to be developed in the future.

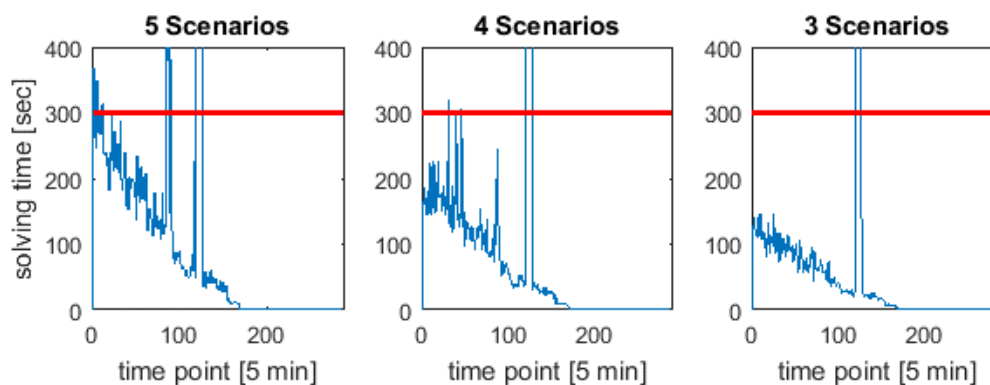


Figure 23: Computation time for different numbers of scenarios (Data 3)

6.4. Warm start / solver change

Since the OP is solved for the same time periods repeatedly with only lightly changed preconditions (updated forecast, SOC different than expected), it is expected to be very useful to use the solution of the previous OP, to re-dimension it for the current problem, and to hand it to the solver as first feasible solution. This user-proposed solution is called warm-start and should already be close to the optimal solution. Thus, the remaining OP period, after having found a solution for the root relaxation, can be spent on trying to find a better solution than the warm start.

Because, as shown in the foregoing sections, the CPLEX optimizer is already able to solve the OP sufficiently well, even when interrupted before having found an optimal solution, the tests were run with the open source solver CBC, in order to determine, whether a warm start could bring the free solver back into competition. In chapter 4, it was shown that CBC was not able to find an optimal solution to the OP of the EMS. However, as demonstrated in section 6.2, it is not necessary to find an optimal solution, since the variance of the objective for several simulations of the same day without a time limit is higher than the degree of optimality lost by deploying a solver time limit. Technically, the time limit has no negative effect on the EMS results in this test case with CPLEX. Therefore, in this section, CBC is tested on its ability to find satisfactory results when a solver time limit and a warm start are deployed.

Remark: Another reason for changing the solver is, that problems arose by combining Matlab and CPLEX. These problems are described in section 7.5.

The results for the three PV generation data sets are shown and compared to the time limited CPLEX simulations in Table 6.



		CPLEX time limit	CBC time limit	CBC time limit & warm start
E_s [MWh]	Data 1	44.369	43.898	43.574
	Data 3	43.004	43.167	43.201
	Data 5	47.558	46.808	46.941

Table 6: Comparison of CBC and CPLEX

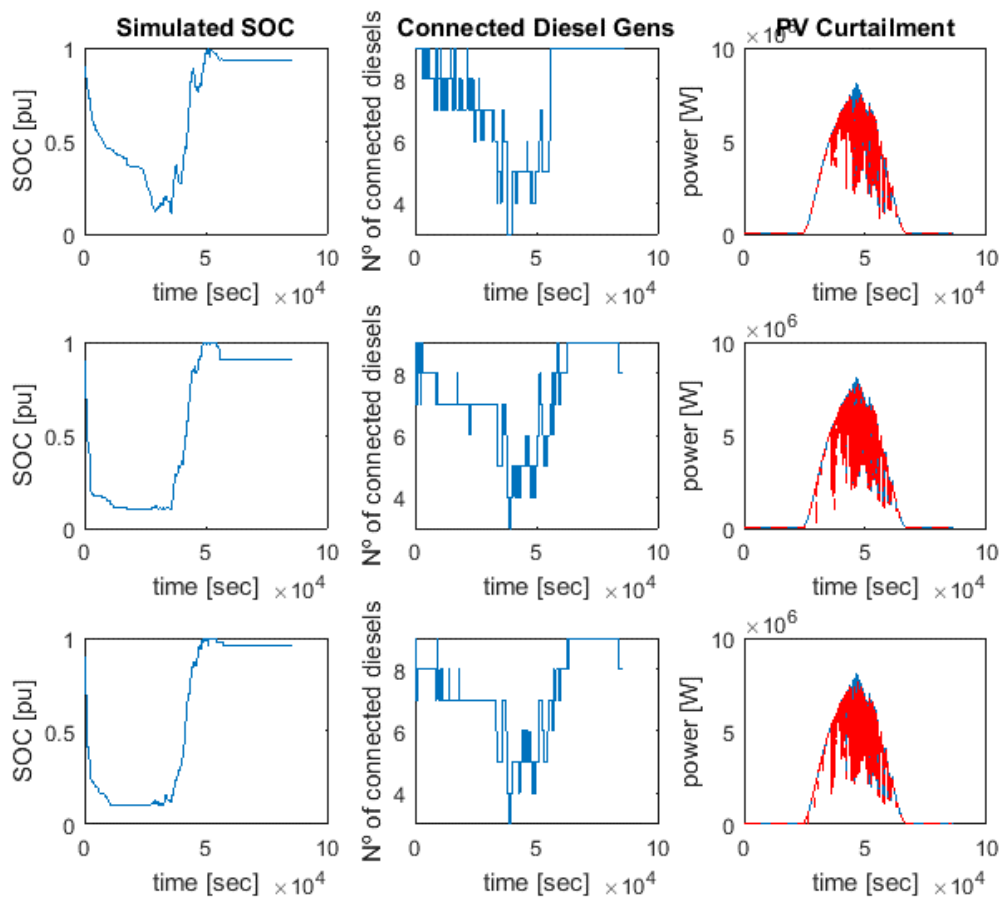


Figure 24: Comparison of time limited CPLEX (top), time limited CBC (middle), and time limited CBC with warm start (bottom) - Data 1

Surprisingly, and somewhat opposing the findings of the quick test in chapter 4, CBC is clearly able to compete with CPLEX for the current test case. In Figure 24, a graphical comparison between the three cases is shown for the three variables controlled by the EMS (exemplary

for Data 1). It can be seen, that the simulations on CBC basis have a less “detailed” behavior than the CPLEX simulation. Nevertheless, the overall behavior can be labeled as acceptable. The warm start seems to slightly increase the quality of the behavior as the SOC trajectory is a bit more similar to the CPLEX case. This has, however, no positive effect on the objective value of the results.

The warm start tested in this section does not result in a significant improvement of the results in opposition to the prediction. If any, it only increases the results slightly. More tests on other systems should be run to evaluate the warm start. Considering the low effort of implementing a warm start, it is worth to implement it anyway, even with low positive effects on the results.

Furthermore, once again it can be demonstrated clearly, that an optimal solution to the OP is not required for a good performance of the EMS. Under the current conditions, other factors have a way stronger impact on the EMS result than the optimality of the solution. This finding is particularly interesting for smaller systems, as CBC becomes more competitive.

6.5. Modelling the problem as a LP instead of a MILP

To solve a MILP, many LP relaxations have to be solved successively (see section 3.2). Therefore, if the problem is reformulated to a LP, the time to solve the problem decreases significantly to the order of magnitude of the LP root relaxation. In case of the first OP execution of the day, the MILP problem took 515.5 seconds to solve, whereas the LP relaxation only took 16.2 seconds to solve on the same machine. Generally speaking (independently of the machine where the OP is solved) that means that the LP can be solved about 30 times faster than the MILP. Therefore, in theory it would extremely increase performance to model only with continuous variables instead of including binary variables. How much the performance would increase exactly, depends on the real LP implementation of the problem (in this case only the binary variables were made continuous which is not a valid model for the EMS). The exact benefit will only be clear when the mathematical model is reformulated.

A very important cost factor is, that, in contrast to the best open-source MILP solver CBC (see chapter 4 and previous section), the best available open-source LP solver (CLP) is very comparable to CPLEX in terms of performance as demonstrated in [13] (when solving a problem to optimality). That means, CLP is able to solve many problems within the same time frame as CPLEX. As a small test, the root relaxation of the EMS problem was solved to optimality with CLP which took 70 seconds on the student’s machine (hardware details given at the start of section 6). Although this is about 4 times slower than CPLEX, it is still very suitable for the EMS application. The objective value of the result is the same for both solvers.

Considering that the license costs for CPLEX lie in the range of several thousands of dollars



per year per machine, the LP has a very big financial advantage over the MILP. Since reformulating the mathematical problem, however, is a very complex task and lies beyond the scope of this thesis, it is only recommended at this point, that the financial and mathematical advantages and disadvantages of reformulating the mathematical model in the future should be evaluated.

6.6. Conclusions

The tests conducted for this thesis project show that the results cannot be significantly improved/changed in terms of objective value of the results by any of the measures tested. In comparison to the fluctuations between several identical simulations of the standard case, the influence of the parameters tested are visible but small. These fluctuations observed for identical simulations most probably result from the random scenario generation and the non-deterministic nature of day-long simulations for the EMS. The solver might find different variables for the same objective value, which can create differences between OP results although the objective value of the OP of both has the same optimal value.

This observation, on the other hand, means that measures to keep the OP solve time within its boundaries, result in no significant reduction of the EMS yield. Be it setting a time limit to the solver, using older results, or adapting the optimization period. Having that in mind, in most cases it will be the cheapest and easiest alternative to set a time limit to the solver. This will not significantly decrease the overall OP results. Only in cases, where the limit is very low compared to the solve time without limit (mean value over the whole day), the results might be heavily affected. This must be tested, when implementing the EMS in a real case.

Another measure that has no risks/costs but was expected to have the potential to improve the results, is to utilize the warm start functionality present in most solvers. To reuse the previous result as first feasible solution to the solve process did not result in the expected benefits. The improvements that resulted from the warm start were only very light (if even observable). Still, considering the low effort of implementing a warm start and no drawbacks, it could make sense to use it. More tests should be conducted on other systems.

One of the most surprising findings is, that the open source solver CBC could generate results, that are comparable to the ones using CPLEX. The realization that the solver does not need to solve to optimality in the EMS case, made it possible to produce acceptable results by deploying a time limit for CBC. This result somewhat corrects the findings in chapter 4, where the solvers were compared merely based on the time to solve (to optimality) of a single OP. This finding could have a significant impact on the costs of the EMS.

Reducing the number of scenarios could, in fact, improve the objective of the results. However,

reducing the number of scenarios negatively impacts the EMS' ability to successfully counteract uncertainties and unpredicted events. Therefore, the following approach is proposed when implementing the EMS in a real case:

1. Choose the acceptable/necessary power reserves (minimum number of connected diesel generators, power margin of diesel generators, battery SOC reserve, battery power reserve, ...)
2. Conduct tests over some months to determine what number of scenarios is sufficient to face unpredictability and to keep the frequency within desired boundaries
3. Choose hardware/solver combination that can solve the OP in acceptable time and that are affordable for the customer. Maybe other factors play a role here like what hardware is already installed on the plant.

It must be recalled here, that the tests conducted in this chapter cannot be regarded as representative due to the limited project time frame. The results are only meant to give qualitative advice on how to best implement the OP in a final implementation that will vary based on the plant size and customer requirements. The final tests will take place, when the EMS is implemented on a power plant.



7. EMS implementation in C++

As a preparation for the final implementation of the EMS, it was decided to implement the system in C++. The background for the C++ decision is rather practical: GPM employs mainly C and C++ developers and the product, once finished, will have to run as a stand-alone application (without Matlab) in the field. Therefore, using C++, the EMS will fit in with other GPM products and there is no need to learn a new programming language. This is particularly important considering, that some parts of the EMS, like the statistical preparation of the forecast data, are still to be implemented in the future for the final application. In this work, the forecasts and scenarios created artificially with the R programming suit (see section 3.1.5) are used. Only the main parts of the EMS were implemented and tested in this section of the thesis project.

Before the OP could be implemented, the target platform was to be determined. GPM uses three different platforms to run software and processes in the power plant:

- IA240 RISC-embedded computer with the following specifications [14]:
 - MOXA ART ARM9 32-bit RISC CPU, 192 MHz
 - 64 MB DDR2 SDRAM
 - Linux 2.6.9 operating system
- UC-8100-ME-T Series - Communication-centric RISC computing platform with the following specifications [15]:
 - ARMv7 Cortex-A8 1000 MHz processor
 - 512 MB DDR3 SDRAM
 - Debian ARM 7 operating system
- ProLiant DL360 Generation9 server platform with the following specifications depending on the specific model [16]:
 - Intel® Xeon® E5-26XXXv3 or v4 processors with 1.7 – 3.6 GHz
 - DDR4 Registered (RDIMM) or Load Reduced (LRDIMM) ranging between 128GB and 3TB
 - Windows Server operating system
 - Servers can be upgraded with hardware depending on the requirements

Both Linux based devices have considerably less power than the desktop computer used in this work to test the EMS performance, which was not able to solve the EMS test case (Cobija) in less than 5 minutes at the beginning of the day. As described and shown in the previous sections, the hardware power is a critical parameter for the effectivity of the EMS. It was



therefore decided to (first) implement the EMS as an application for the GPM micro servers that are the most powerful devices deployed on the power plants monitored by GPM. This way, the EMS is likely to solve to optimality in every OP execution of the day and thus, maximize the EMS utility. It must be recalled here, that the complexity of the OP varies widely with the size of the system among other parameters (see section 6). It is not possible to take final decisions about the hardware needed in this early stage of the EMS implementation. Therefore, the EMS was implemented in Microsoft Visual Studio (VS) as a C++ Windows application. Another argument that confirmed this decision was the absence of an academic CPLEX license for the deployed Linux system.

In the following sections, the development process and design decisions are presented more in detail.

7.1. Core EMS and testing environment

As long as the EMS is not controlling real converters and diesel generator sets, the system needs to be tested in an artificial testing environment. This is closely related to the statistical processing of input data which depends on whether there are real forecasts in a real test case used, or if the tests are executed in an artificial environment in which case the forecasts need to be generated artificially. The latter case is how the EMS prototype is implemented in Matlab and GAMS (i.e. Julia). Since, in the foreseeable future, the EMS will not be tested on a real power plant, it was decided to only implement the core EMS in C++ and keep the forecast/scenario part in R for now. This makes sense especially considering the fact that large parts of the statistical scripts implemented in R actually create forecast simulations by perturbing simulation data; a part that will not be necessary anymore, once the EMS works with real forecasts. Another large part of the statistical scripts examine the statistical properties of the generation and consumption data of the power plant and only need to be executed once when installing the EMS on the plant (see sections 3.1.2 and 3.1.3). The only part that will need to be added to the C++ implementation in the future, is the scenario generation from the forecast data. This will not be very complicated when the structure of the forecast data is known.

In Figure 25, the whole EMS project is structured into parts in the same manner as the program scripts and pieces are structured. The statistical part is implemented in R and marked blue in the figure. The main part is implemented in Julia, using the CPLEX solver, and marked in green. This part was originally implemented in GAMS instead of Julia in the first prototype version of the EMS. The body of the simulation environment is implemented in Matlab/Simulink and marked in orange. When testing the EMS in simulations, the Matlab scripts and models coordinate the program execution.

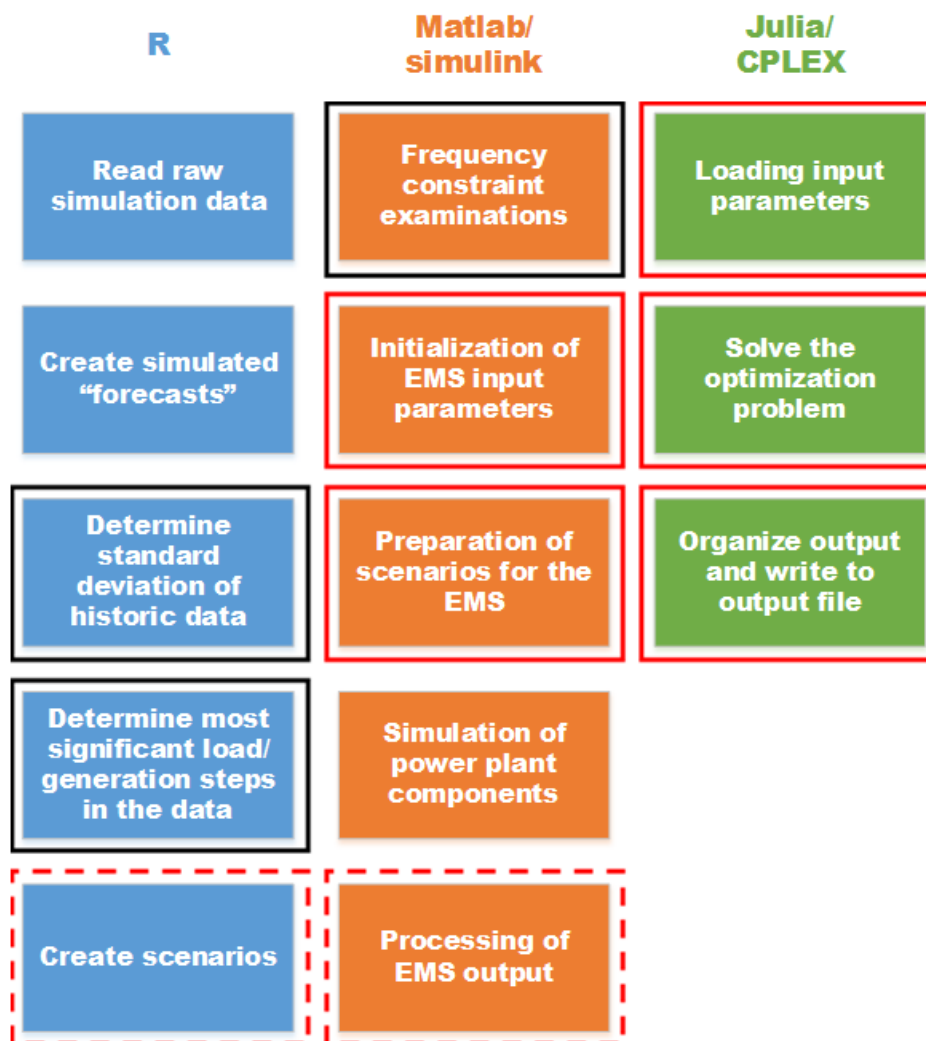


Figure 25: Core EMS (marked red) and other parts of the EMS project

For a stand-alone version of the EMS, all the red framed rectangles need to be implemented in the program. The black framed rectangles are only run once, when the EMS is set up on a specific power plant. Therefore, the parts do not need to be integrated into the EMS.exe program and are not run on the site. Thus, the implementation in R/Matlab/Simulink can be kept as it exists already in the first prototype (see sections 3.1.2, 3.1.3, and 3.1.5). The rectangles without frame will not be necessary anymore, once the EMS works independently of a simulation. All the red framed rectangles together form the core EMS. The two dashed boxes are currently not implemented in C++ yet, because of the following reasons:

- The implementation of the scenario creation from forecast data mainly depends on the format and structure of the forecast data. This means, which file types (or data base structure), time resolution, update period, etc. is used. At the time of this thesis project, however, the format is not clear yet.



- The EMS output will most probably be sent to the converters via Modbus protocol as implemented and shown already in section 5.2.2. To properly implement the communication, the interface should be developed with specific knowledge about the communication capabilities of the converters on the plant. Therefore, it is more practical to develop the interface once, the EMS is installed on a plant. It can be added that the interface used in this project to communicate with Matlab (see section 7.2) can be used as part of the final implementation. Another small program would be needed to send the comma-separated setpoints to the converters every 30 seconds.

In this thesis project, the C++ implementation of the EMS is tested in the Matlab/Simulink simulation environment. This means that Matlab calls the program EMS.exe once every OP execution period. In contrast to the Julia implementation described in section 5, the EMS.exe program loads all the required input data autonomously from xml and csv files (see sections 7.2 and 7.4). Matlab only times the execution and loads the input and output data from and into Simulink. This means that in the final implementation, the program EMS.exe would need to be placed in a timed loop to be executed every 5 minutes. Another small program would be necessary, as mentioned above, to send the calculated setpoints to the converters every 30 seconds via Modbus. A third small program would need to update the forecast data in the forecast folder whenever there are new forecasts available.

7.2. Input and output data format

The input and output data of the program is organized in text files of different formats. The generation and consumption scenarios are saved as csv-files. In the current test phase the format is as follows:

Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
30-secondly data for 24 hours – 2880 data points	30-secondly data for 24 hours – 2880 data points	30-secondly data for 24 hours – 2880 data points	30-secondly data for 24 hours – 2880 data points	30-secondly data for 24 hours – 2880 data points

Table 7: Schematic scenario input format in current test implementation

There is one separate csv-file per scenario for both generation and consumption. That means that there are ten files for one complete update. Every file contains 2880 comma-separated

values; one value every 30 seconds for 24 hours. In this implementation, the scenarios are only updated once a day and a maximum number of 5 scenarios is available in the input data. Therefore, the scenarios, as they are now, have rather a dummy function than to really optimize the EMS performance. In the final implementation, the forecast and scenario preparation needs to be improved. This means that the forecast data should be updated e.g. every half an hour with high quality forecasts that might also include cloud movements. These forecasts are then perturbed with the standard deviation of historical data to create several scenarios (see section 3.1.2).

The main input file is called `usrInput.xml` and contains all the necessary parameters (apart from the scenarios) for the optimization. It is subdivided in three groups of parameters:

- `<regularUpdate>` contains the two parameters `<timeIndex>` with the index of the current OP execution period and `<SOCini>` with the current SOC of the battery. These two values are updated before the start of each OP period.
- `<userData>` contains the setup parameters for the optimization: scaling factor of the simulation data, length of OP execution period in minutes, time resolution of forecast data in minutes, number of scenarios to be considered, and solver time limit information in seconds which is divided into an initial time limit for the first OP execution of the day and a time limit offset for the remaining OP execution which sets the number of seconds before the start of the next OP execution period when the current optimization needs to finish.
- `<OPInputParameters>` contains all the parameters that appear as variable limits, constraint parameters, or parameters of the objective function in the mathematical formulation. In the Matlab prototype of the EMS, these values are set in the Matlab script that also initialized an OP execution. Examples of these parameters are the coefficients of the frequency constraint ϕ_0 to ϕ_3 (see section 3.1.3), the number of diesel generators and their minimum and maximum output power, the battery efficiency, or the minimum allowable system frequency.

Both the “`usrInput.xml`” file and the scenario files are imported at the beginning of every OP execution. This leaves the possibility to modify every piece of data in these files between the runs. Examples are forecast updates or the `<regularUpdate>` parameters.

The results are organized in four files that the EMS outputs after every OP execution. Three of the files are csv-files and contain the setpoints that are updated regularly. The battery power file “`Pbat.csv`” and the maximum PV power file “`PpvMax.csv`” are structured as follows:



First setpoint after current OP execution	Setpoint 30 seconds after current OP execution	...	Last setpoint before next OP execution results are available
...
First setpoint after last OP execution of current day	Setpoint 30 seconds after last OP execution of current day	...	Last setpoint of current day

Table 8: Schematic setpoint output format

The rows contain all the setpoints between two OP executions separated by commas. Every row is separated by a line break. That means that the file contains setpoints for all remaining 30 second periods of the day. For example, if the OP is solved every 5 minutes, there are 10 columns and 288 rows after the 0th OP execution of the day (which takes place during the last 5 minutes of the previous day). Every five minutes, a new table is generated with one row less than the previous table. In normal operation, it would be sufficient to only store the first row after which a new first row would be calculated. However, designed as shown in Table 8, if one OP fails, the setpoints calculated in the previous run (second row) can be used.

For the number of diesels in the file “nDieselON.csv”, the table only has one column, as this setpoint is only updated once per OP execution period. In case that the OP execution period is 5 minutes, the file contains 288 values separated by line breaks after the 0th OP execution.

This output data format is the one developed at CITCEA, as already implemented in the Matlab/GAMS prototype, with the difference that it is now implemented with csv-files instead of mat-files.

The fourth output “metadata.xml” file is an xml-file and contains information about the OP process. The four fields that are currently implemented are

- <SolveStatus>: Contains the CPLEX status of the optimization, e.g. “Optimal” when solved to optimality, or “AbortTimeLim” when the solution process was aborted due to a user defined time limit.
- <ElapsedTime>: Measures the absolute time of the solving process
- <AccumulatedCPUtime>: Adds up all the computation time for every core
- <ObjectiveValue>: The objective value at the solution

7.3. Visual Studio settings

For the correct functioning of the compilation with visual studio, some preparations need to be made. The first step when starting the project, is to create a new C++ console application. Then the project's property pages need to be opened and in the configuration manager, the configuration needs to be set to "release" and the platform to x64. In general, the target platform needs to be the same for the precompiled CPLEX library and the code written by the user. Another important restriction is, that the libraries are available precompiled for different VS toolsets. In case of the CPLEX version used in this part of the project (Version 12.7), there are libraries available for VS versions 13 and 15. Since in this project, VS version 17 was used, the "Platform Toolset" field on the general property page needs to be set to Visual Studio 2015 (v140). Then, the precompiled CPLEX libraries in the folder "x64_windows_vs2015" can be used by the linker.

Under C/C++ → General, the two include directories for "concert" and "cplex" in the CPLEX program folder need to be filled in. That lets VS find the CPLEX header files when writing `#include <ilcplex/ilcplex.h>` at the beginning of the main.cpp script. Under C/C++ → Preprocessor, "IL_STD" needs to be added to the preprocessor definitions.

As a last step, the linker needs to be given the information which additional libraries are used and where they are located on the machine. For this purpose, the two full paths to the library directories of CPLEX and Concert need to be inserted in the "Additional Library Directories" field under Linker → General. In this case these are

- ...\\CPLEX_Studio1271\\concert\\lib\\x64_windows_vs2015\\stat_mda
- ...\\CPLEX_Studio1271\\cplex\\lib\\x64_windows_vs2015\\stat_mda

Then, the three lib files "cplex1271.lib", "concert.lib", and "ilcplex.lib" need to be specified in the "Additional Dependencies" field under Linker → Input.

When building CPLEX applications in general, it is crucial that there are library files available for the correct platform (e.g. 64bit Windows), and that a VS version is used, that is compatible with the toolset with which the libraries were built. If this is not given, the linker will not be able to link the CPLEX libraries to the rest of the program. When porting the application to another device, the Visual C++ Redistributable package for the corresponding VS toolset version (the one by which the application was build) must be installed and the correct CPLEX and Concert libraries need to be available and reachable by adding them to the system environment variable when installing CPLEX.



7.4. Program structure

The way how the program EMS.exe is structured is due to how the development process will proceed. In the main function, the parameters are set, the model is created, the OP is solved and the output is written to files. The forecast loading and processing is separated from the main function. All the associated functions are defined in the file "LoadForecasts.cpp". In the main function, the forecast data can be loaded by creating a forecast object and applying the "loadForecasts" function on it. Then the data is queried by means of the function "getForecasts". The reason for this is that the forecasts loading process and the scenario generation will need to be revised, once real forecast data is available. This way, only the "LoadForecasts.cpp" file will need to be changed. The remaining source files contain the classes and functions for the csv- and xml-file processing. In the following, the source files are presented more in detail and the sequence of the EMS.exe is explained.

7.4.1. LoadForecasts.cpp

In the file "LoadForecasts.cpp", the functions of the class "Forecast" are defined. This class has two private members and two public functions in the current implementation. The two members are the 2-dimensional vector matrices "Consumption" and "Generation", that contain all the scenario data in columns.

The first function in the class is "loadForecasts". Its argument structure is shown in the following:

```
void loadForecasts(const int scen, const int scale, const int timeIndex, const int tIntra);
```

It fills the two members "Consumption" and "Generation" with scenario data stored as csv-files in the folder "Forecasts". The function first loops through all the files called "consum"-scenario index-".csv" (the highest considered scenario index is "scen"), reads all the values and stores them in columns. The exact process of reading csv-files is described in section 7.4.2. This process is repeated for the files named "irradiat"-scenario index-".csv". After that, all the values of the forecast data that lie in the past (as seen from the current OP execution) are deleted from the matrices. These values to delete are the "timeIndex" * "tIntra" values at the beginning of the vector. This step is crucial for the functioning of the EMS. The forecast data needs to contain values for all the considered scenarios, from the present (as seen from the current EMS execution) until the end of the day, and for every 30 seconds. In case that the forecasts are updated every 30 minutes, it is still necessary to erase data that lies in the past, in 5 out of 6 EMS executions within these 30 minutes (in case of an OP execution period of 5 minutes). Only if the forecast update happens more frequently than the OP execution period, this step is not necessary anymore. As a last step, all negative values and zeros are deleted from

“Generation” and the complete vector is scaled with “scale” to better match the consumption data. The scaling of generation data only serves for research purposes and will not be part of a final implementation of the EMS.

The other function queries the data in the two private members of “Forecast”. The function expects one of the two selector strings “Consumption” or “Generation”. Corresponding to the selector string, either the vector matrix “Generation” or “Consumption” is returned as output parameter. For other input arguments, the function throws an `invalid_argument` exception.

7.4.2. Other external source files

Two other source files and corresponding header files exist in the project that deal with the input and output data in csv- and xml-format. “CSVread.cpp” is a small file created to read the values separated by commas in the forecast input files. In the header file, a class “CSVrow” is defined that has the private member “m_data” with type `std::vector<std::string>`. The class has three public functions:

- The operator `[]` is overloaded to return indexed elements in the class member “m_data”. This way, the read data from the csv-file can be queried from the private member.
- The function “size()” returns the length of “m_data”. This function can be used in combination with the `[]` operator to return the whole vector.
- “readNextRow” fills the vector “m_data” with the data of one row of the csv file. The values separated by commas are read and saved in the vector as strings. If this method is looped, a whole csv file can be read row by row.

Furthermore, outside of the class “CSVrow”, the extraction operator `>>` is overloaded for “CSVrow” objects. Now, this operator can be used to extract data from input streams (files or other mediums) into a “CSVrow” object. The operator has the same effect as calling the function “readNextRow”, however the terminology is more logical to use with C++ streams.

The other file included in the project is auxiliary software to parse xml files. “Pugixml” is one of the most frequently used xml-parsers. Its advantages, in this case, are the ease of use and its limitation on basic xml-file parsing operations. The library and more information can be downloaded from [17]. The library is just included in the project as cpp-file with the corresponding header files “pugiconfig.hpp” and “pugixml.hpp”. Then an exemplary numerical value “fMin” that is saved in the input file “usrInput.xml” under the tag path “EMSdata” -> “OPInputParameters” -> “fMin” can be extracted as double as shown below:



```
pugi::xml_document doc;  
doc.load_file("usrInput.xml");  
  
const double fMin =  
doc.child("EMSdata").child("OPInputParameters").child("fMin").text().as_double();
```

7.4.3. Main function

The main function of the EMS.exe application is structured into five parts:

1. Allocation of EMS parameters with data from the file “usrInput.xml”
2. Loading the forecast data
3. Optimization model construction
4. Solving the model
5. Saving the results in output files

In the first part, the file “usrInput.xml” is loaded by means of the xml parser “pugixml”, which is a widely used light-weight parser for basic operations with xml files [17]. First, an instance of the class “xml_document” is initialized which is then populated with the content of “usrInput.xml”. In the next step, the values are queried as integers or doubles and assigned to constants, that are then used later in the program, either as mathematical parameters, indices, scaling factors, or CPLEX option parameters.

In the second part, the forecast data is loaded (scenario creation in future EMS version). First a forecast object is declared, then the function “loadForecasts” is applied on it to populate the object as described in section 7.4.1.

All the remaining program parts utilize ILOG Concert Technology. This package contains a set of modelling objects available for C++ among other languages, that make it possible to embed CPLEX into a C++ application. The frame for the three last parts of the program is the declaration of an environment object “env” at the beginning and the destruction of it at the end of the program. The environment object contains all the other concert technology objects used within the program that are divided into modelling objects and CPLEX objects (see section 3.4.2). In case of the EMS, the objects are

- variable array of two, three, and four dimensions
- constraints
- an objective function
- an IloCplex-object for the OP solve
- other small objects like timer, expressions, iterators, etc.

All these objects are associated with the environment object. Calling the function “env.end()” at the end of the program deallocates all the space used by the Concert objects. Apart from the two lines where “env” is created and destroyed, the three last program parts are enclosed in a try-catch expression to deal with exceptions that are thrown, in particular “IloExceptions” that originate from Concert technology.

First in the model construction step, the different variable containers are defined. All the variables in the mathematical model of the EMS are organized in 2- to 4-dimensional arrays. Since in Concert technology, only 1-dimensional “IloNumVarArrays” exist, the necessary arrays are defined as in the following:

```
typedef IloArray<IloNumVarArray> NumVarMatrix;
typedef IloArray<NumVarMatrix> NumVar3Matrix;
typedef IloArray<NumVar3Matrix> NumVar4Matrix;
```

In the next step, every single element in the arrays needs to be initialized. That means that arrays and sub-arrays are given a size and are connected to the environment object “env”; variables are given bounds, a type, and a connection to “env” as exemplary shown in the following:

```
Xcar[i][j] = IloNumVar(env, 0.0, 1.0, ILOBOOL);
```

The first parameter is mandatory, because otherwise, only an empty pointer is created. The second and third parameters are the lower and upper bound on the variable which are negative infinity and positive infinity by default. The fourth parameter is the variable type which is ILOFLOAT (Floating point number in Concert technology) by default.

After having initialized every single element in the variable arrays, constraints are added to an “IloModel” object that is declared first, meaning to connect it to the environment object. All the variables created in the first step are only connected to the model through the constraints. The syntax for adding a constraint to the model is “model.add(IloExpression);”. As an example, in the following the creation of the set of SOC constraints is shown:

```
for (IloInt i = 0; i < tEMS; i++)
{
    for (IloInt j = 0; j < tIntra; j++)
    {
        if (i == 0 && j == 0)
        {
            model.add(SOCbat[i][j] == SOCini - Pbat[i][j] / batCap);
        }
        else if (i > 0 && j == 0)
        {
            model.add(SOCbat[i][j] == SOCbat[i-1][tIntra-1] - Pbat[i][j] / batCap);
        }
        else
        {

```




```

        model.add(SOCbat[i][j] == SOCbat[i][j-1] - Pbat[i][j] / batCap);
    }
}

```

“tEMS”, “tIntra”, “SOCini”, and “batCap” are parameters imported from the input xml-file in the first part of the main function (see above), “SOCbat” and “Pbat” are 2-dimensional variable arrays.

In the last step of the model formulation, the objective function is composed of the sums of battery power and PV power, and the parameters “scen” and “etaBat” from the input xml file. First the sums of all the elements in the variable arrays “PbatCar”, “PbatDesc”, and “Ppv” are created in a forward loop as *IloNumExpressions* “totalPpv”, “totalPbatCar”, and “totalPbatDesc”, then the objective function is added to the model as shown below:

```

model.add(IloMaximize(env, totalPpv - scen * (1 - etaBat) * (totalPbatCar +
totalPbatDesc)));

```

The fourth part in the main function is to solve the model. For this purpose, an *IloCplex* object called “cplex” is declared providing the *IloModel* object as input parameter. In this format, the model can be solved and all kinds of aspects of the solution can be queried. Before solving the model, the input parameters are set as shown below for the CPLEX time limit:

```

cplex.setParam(IloCplex::TiLim, timeLimit);

```

The first input parameter to “setParam” is the name of the CPLEX parameter that is to be modified, the second provides the corresponding value. The value “timeLimit” is calculated from the input parameters to the EMS as follows:

```

IloNum timeLimit = tIntra * 30 - tilimOffset;

```

After that, as a last step before solving the model, the number of variables/columns and constraints/rows is printed on the console to check the correct composition of the model at every execution of the EMS.

Then the CPLEX solver is called to solve the model with the command

```

cplex.solve();

```

This part takes the main portion of the program execution time. The accumulated CPU time and the real computation time are measured with timers. During the execution, the solver outputs information about configurations, solve strategies, and events that happened, like number and type of applied cuts, best primal and dual continuous solutions, best integer

solution, gap between the two in percent, elapsed time of different steps in the solving process (e.g. root relaxation solution time), number of nodes in the search tree, and marks when a node was fathomed (integer, infeasible, cut-off) (see Figure 22). When the problem is solved, a summary of the solve process is given before the console window closes.

As a last step, the (most important) results contained in the IloCplex object are extracted and written to text files by means of C++ output file streams. This C++ class allows to write to files in a very straight forward way. It makes use of the concept of Streams in C++ that lets the user write different types of data to all different kinds of storage media. First, the solve status, solve times, and objective value are written to a text file in the xml-format between the corresponding tags. Then the required setpoint data is written to csv-files by separating values in different columns by commas and rows by line breaks. The exact format is explained in section 7.2. Obviously, the exported data in the current implementation is only the minimum required data. It might be necessary to export more data at a later point, that is, to make modifications on this part of the program. As an example, how the cplex class of the Concert technology provides functions to extract data from the object, the syntax to extract the objective value of the solution is displayed in the following:

```
cplex.getObjValue()
```

At the end of the main function, the exceptions are caught (separated catch expression for IloExceptions and other exceptions). Then the environment object “env” and all other objects associated with it is deallocated by calling the end function.

7.5. Test

The developed EMS prototype is able to reproduce the desired results. The EMS yield cannot be compared directly to the results in chapter 6, because the generation scenarios used in the C++ prototype are pre-generated and only used as a space holder for a later scenario implementation. The behavior of the EMS is illustrated in Figure 26 (exemplary for Data 1 and an OP execution period of 10 minutes). Just as in the original prototype, the SOC of the battery decreases during the early morning hours to make space for preferable much PV power during the day that cannot be directly consumed. The number of diesel generators decreases as the PV power increases. In the morning, it can be decreased earlier due to the additional battery power. The curtailed PV power (red curve) is kept as high as possible. This shows, that the EMS was implemented correctly.



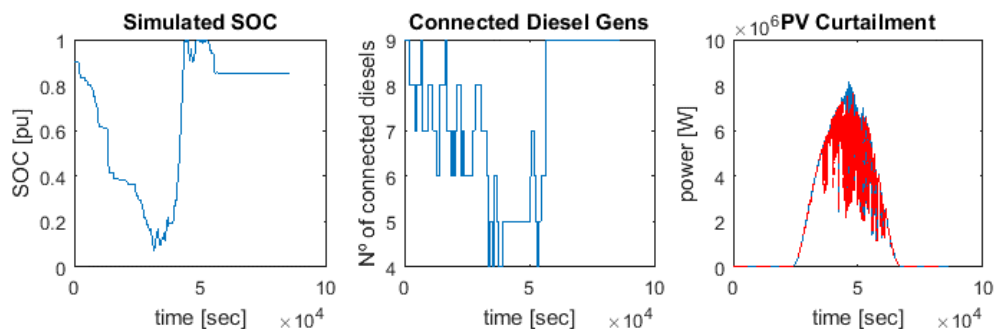


Figure 26: Behavior of the system controlled by the C++ EMS

A problem that occurred during the tests seems to be due to an incompatibility between CPLEX and Matlab/Simulink when being run in parallel. In rare and unpredictable occasions, the EMS execution in combination with Matlab produces a segmentation fault in memory, causing the program to crash. In one case, this fault even resulted in a blue screen and a complete shutdown of the computer. The problem seems to be connected to a bug in CPLEX mentioned in [18], even though the described setup is not the same as in this project. It could not be determined exactly what causes the problem during the course of this thesis. This issue raises new questions about the solver choice. It may be necessary, to switch to another solver in the future if the problem persists. The findings about CBC in chapter 6 also suggest using CBC, at least for smaller power plants.

Remark: The problem occurs in both applications containing CPLEX – C++ and JuMP, only in combination with Matlab.

8. Costs of the project

One of the objectives of the project is, to study its economic feasibility. For this purpose, it was decided to compare the following three (main) cases:

1. The power demand is met only by the diesel generator set.
2. The hybrid power plant consists of a PV generator feeding into the grid when possible and a diesel generator set with all generators always running.
3. The basic setup of this thesis: diesel generator set, PV generator, and a battery storage are coordinated by the EMS in order to optimize the system's solar power utilization and battery losses.

On the basis of these cases, a cost comparison is conducted over a project time frame of 10 years. In this comparison, investment costs for different components as well as fixed and variable running costs of the cases are summed up. Assuming a discount rate of 5%, the costs are accumulated and actualized and can be compared.

The dimensions chosen for the system components in this cost analysis are the same as in the previous parts: a battery storage with a capacity of 1120kWh and input/output power of 2200kW and a diesel generator set of 9 generators with an output power between 400kW and 1200kW each. The assumed demand for the first year is the integrated consumption profile multiplied by 365 days. In the following years, this sum increases by 3% per year. The diesel generation in the first year was estimated to be the mean value of the integrals of the three different simulations (Data1, Data3, Data5) for one case (Case 1, Case 2, or Case 3) multiplied by 365 days. The formulas are given in the following:

$$E_{demand,year1} = \int P_{demand} * 365 \quad (13)$$

$$E_{Dies,year1} = \frac{\int P_{Dies1} + \int P_{Dies3} + \int P_{Dies5}}{3} * 365 \quad (14)$$

By means of these formulas, the energy sum of the first year is calculated. It must be noted that the generated diesel energy varies with the amount of PV power used by the system which in turn varies with the examined case. In the first case, the power generated by the diesel generator set is identical with the demand. Analogously, the energy generated by the generator set increases with the same offset to the demand as in the first year, since the PV generator does not produce more energy and the difference has to be balanced with diesel power. Obviously, this assumption is rather conservative, since the PV power is not fully used during



midday. Thus actually, the PV power could be increased a little over the following years if calculating in a less conservative style.

In case of this comparison, the generators are assumed to be already installed at the beginning of the project, and no investment for the generator set is necessary. This also makes sense, as different cases are compared that all have the same number of generators, thus the investment costs would be cancelled out when subtracting the different cases (see Table 9). As maintenance costs, 5000€/machine and annum are stated for generators of this power class in [19]. Multiplied by the number of generators, these are the only running fixed costs in the first case. The variable costs are calculated by dividing the fuel price (0.64€/l in the first year, then increasing by 2.5% per annum [20] and [21]) by an energy content of 4kWh/l [22]. Then for every case, this price is multiplied by the energy generated by the generator set.

In the second and third case, the PV investment costs are estimated to be 2M€/MWp installed PV power [23]. In the case of this project, 8MWp are installed, resulting in a total investment cost of 16M€ for the PV generator. Additionally, 1.5% of the investment costs per year are necessary for its operation.

In the third case, additionally to the PV generator, a battery storage and the EMS consisting of a workstation PC and the CPLEX license need to be purchased and operated. For the battery, an investment of 400€/kWh for the cells and 100€/kW for the converter is estimated [24], resulting in a total investment sum of 668k€. The fixed costs of the battery operation are 1.5% of the total initial investment, considering the price 8.5€/kWh per year [25]. Finally, the costs of the workstation (1000€ investment) and the CPLEX license (60000€/year) are added to complete the picture. For the CPLEX license, a four-core processor needing licensing for 100 processor value units at a cost of 150€ per year and processor value unit is chosen as a basis for calculation [26].

	Accumulated actualized costs (€)	Difference (€)
Only Diesels	115,044,513.26	0
Diesels + PV, no EMS	113,536,347.30	1,508,165.96
Diesels + PV + Battery + EMS	110,950,829.12	2,585,518.17

Table 9: Results of cost comparison

In Table 9, the accumulated and actualized costs are displayed for the three cases over the project time frame of 10 years. The total costs range between 111M€ and 115M€. Only adding a PV generator to the system decreases the costs by about 1.51M€. However, adding the EMS with a battery storage saves another 2.59M€ resulting in a saving of 4.10M€ in total compared to the first case with only Diesel generators. It is clearly visible that the EMS is able to significantly lower the costs of a hybrid power plant, even if a commercial solver is deployed.

Replacing the solver by an open source version would not have a large effect on the overall results, especially considering that it would very likely reduce the overall utility of the EMS (depending on the size of the OP). The costs of the license over ten years is 0.66M€. This acknowledges one of the conclusions of chapter 6 that it must be decided on a case by case basis which solver to choose. The larger the system and the harder it is to solve, the more feasible the deployment of a commercial solver becomes.

The spreadsheets of the detailed cost analysis are attached to this thesis in the Appendix.



Conclusions

For this thesis, a new EMS prototype on the basis of Julia/JuMP was developed in order to be able to run qualitative tests on the EMS behavior without a GAMS license. The tests focused on possibilities to reduce the computation time for cases, when the OP cannot be solved to optimality within the standard OP execution period. Furthermore, a first implementation in C++ based on the IBM Concert technology was created to serve as a “first draft” for further development. In the last chapter, a quick economic analysis based on an accumulated cost comparison was conducted to show the economic feasibility of the examined EMS. In the beginning of the thesis, the project foundations are presented, introducing the reader to the EMS project, relevant optimization theory, and deployed modelling software.

By implementing the EMS OP in Julia/JuMP, the solver flexibility provided by GAMS could be maintained. At the same time, since it is a real programming language, developer freedom and simplicity of the overall system could be increased. The Matlab/Julia interface is significantly simplified compared to the original Matlab/GAMS interface. The raw structure of the original prototype was maintained. Furthermore, an EMS version for emulations with real converters was translated to Julia/JuMP, making it possible to run more accurate tests in the future without a GAMS license.

The EMS tests conducted in this thesis resulted in the finding that the differences in EMS yield induced by randomly generated scenarios and other non-deterministic components in the simulations are generally larger than the differences resulting from different EMS setups. This means in return, that different measures to make the EMS solve the OP in time can be mainly evaluated based on their ability to do exactly that, as the EMS yield is never significantly decreased. The most effective measures were found to be setting a time limit in the solver, if necessary in combination with extending the OP execution period. The last OP executions of the day could be skipped (using an earlier result for the remaining setpoint updates of the day), in order to have more time to solve the first OP of the next day to optimality to use it as a warm start for the coming OP execution periods and to thereby accelerate the EMS even more.

A very important finding of this thesis project is that, because of the large result variance, the OP is not required to be solved to optimality. Given that, the solving process can be abbreviated significantly as the most time-consuming part of the OP solution is to find the optimal solution after rather good feasible solutions have already been found. This finding brings the open source solver back into discussion, which can save a significant amount of license costs and is particularly interesting for OPs that are smaller than the example case “Cobija” of this thesis.

The “first draft” of the EMS in C++ is able to reproduce the EMS behavior that was expected



from the experience with the research prototypes. Some space holders are included in the code, in order to facilitate the implementation of other EMS parts like forecast processing and scenario generation. In preparation of the development, the “core EMS” was defined in order to clearly distinguish the project parts that only serve as test infrastructure, and the parts that will be part of the product. Moreover, practical descriptions of the implementation process are given.

The accumulated cost analysis conducted in the last chapter of this thesis acknowledges the economic feasibility of the EMS and justifies further project development. It could be shown that the project costs of a hybrid power plant could be significantly reduced by deploying an EMS despite the additional investment and fixed costs that the EMS and a battery storage system add to the cash flow. However, for a project of the size of the “Cobija” example, the solver license costs are comparably small and it is not advisable to use an open source solver. This situation may change for other cases.

It has to be repeated here, that the findings are qualitative and are intended to give orientation when implementing the project. The exact layout and solver choice has to be determined on a case by case basis depending on the specific project properties. One important aspect of this case by case difference, is the number of necessary scenarios that heavily depends on the uncertainty (i.e. quality of available forecast data), the admissible energy and power reserves in the system, and the desired frequency tolerance. As this thesis demonstrates that more scenarios decrease the result quality and significantly increase the computational effort, preferably few scenarios should be deployed, while, however, using enough scenarios to make the system robust enough to face unpredicted events.

This leads to an important future subproject. The development of the forecast processing and scenario generation algorithms should include another optimization problem that helps the project developers to determine the optimal number of scenarios for a specific EMS project.

This is closely related to the natural follow up to this thesis project that targets the further development of the C++ EMS. This includes the improvement of existing code, the implementation of a sophisticated scenario/forecast functionality, and an improved output data processing part, depending on the enclosing infrastructure (e.g. transmission of setpoints via Modbus). A particularly important development step will be the transformation of the “1-day-system” towards an EMS that runs over several days, mastering the transition from one day to the next. However, these follow up steps cannot replace a project dependent final development step to tailor the system for a specific power plant.

Additionally, some further advices and ideas are presented in the following that result from this thesis project.

It was briefly shown in this thesis, that a LP has clear advantages over a MILP of the same size in terms of computational effort and thus, costs. Therefore, it could be re-evaluated whether it is possible to reformulate the problem as LP.

Furthermore, it would be useful to develop another C++ program on the basis of the CBC C++ interface for cases where an open source solver is sufficient for the task. The program developed in chapter 7 of this thesis can serve as a model as the two codes will be very similar. However, due to the fact that the C++ interfaces are solver specific (different classes and objects), a translation is necessary.

Crashes that occurred during the tests conducted in chapter 6 and 7, when deploying the combination of Matlab/Simulink and CPLEX, reveal a worrying problem of CPLEX. The error is due to a segmentation fault, randomly occurring when run in parallel with Matlab and therefore particularly hard to reproduce or track down. The fact that the CPLEX source code is not accessible, even aggravate the trouble shooting. As this bug makes it very hard to test the EMS in Matlab simulations, it should be taken into consideration to switch to a competing product like Gurobi or MOSEK as these solvers have a better reputation in this respect and especially Gurobi is generally known for solving bugs faster than IBM for CPLEX [18]. This would, however, mean to implement the problem in another C++ API. Moreover, the license has to be purchased. Nevertheless, as a cost comparison between commercial solvers is advisable anyway, this might not add extra work to the project. If CPLEX is found to be the most economic solver choice, it will be necessary to spend more time determining the bug's origin, and to eliminate (or avoid) it. At least, if more simulation testing needs to be conducted.

Finally, one suggestion for future improvement is to change the objective function of the EMS to optimize the operational costs (or at least fuel expenses) instead of PV power use and battery losses. As the final objective of every power plant owner is to optimize the profit of the power plant, this seems to be a more direct approach to manage the system components. It has to be taken into consideration that optimizing the fuel expenses will change the OP formulation and might increase the problem size as the relationship between diesel generator output power and fuel expenses needs to be modelled. This would mainly improve the battery schedule, causing the battery to discharge, when the demand is high and little PV power is available.



References

Remark: The work presented in section 3.1 of this project was already existent when starting the thesis project and the thesis builds upon that work. It is, however, not possible to cite publications, because the reports are created exclusively for GPM and are confidential. So far, there exist no publications on the described project phases. Responsible for the work in section 3.1 are the following members of CITCEA research institute: Andreu Vidal, Monica Aragüés, Eduard Bullich, Guillem Vinyals, and Oriol Gomis.

- [1] A. SOBU; G. WU, *Optimal operation planning method for isolated micro grid considering uncertainties of renewable power generations and load demand*, IEEE PES Innovative Smart Grid Technologies, 2012, p. 1-6.
- [2] G. C. LAZAROIU; V. DUMBRAVA; G. BALABAN; M. LONGO; D. ZANINELLI, *Stochastic optimization of microgrids with renewable and storage energy systems*, Environment and Electrical Engineering (EEEIC), 2016, p. 1-5.
- [3] GUROBI OPTIMIZATION, *Mixed-Integer Programming (MIP) - A Primer on the Basics*, [Online]. Available: <http://www.gurobi.com/resources/getting-started/mip-basics>. [Accessed 28 02 2017].
- [4] IBM, *IBM Knowledge Center*, [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.1/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html. [Accessed 03 03 2017].
- [5] D. G. LUENBERGER; Y. YE, *Linear and Nonlinear Programming*, Stanford: Springer, 2008, p. 49-123.
- [6] *Simplex Algorithm - Wikipedia Article*, [Online]. Available: https://en.wikipedia.org/wiki/Simplex_algorithm. [Accessed 13 03 2017].
- [7] S. ALBERT, *Solving Mixed Integer Linear Programs Using Branch and Cut Algorithm*, Masters of Mathematics, North Carolina State University, 2006.
- [8] I. DUNNING; J. HUCHETTE; M. LUBIN, *JuMP: A Modelling Language for Mathematical Optimization*, Ithaca (New York), 2016.



- [9] IBM, *Concert Technology for C++ users*, [Online]. Available: https://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.3/ilog.odms.cplex.help/CPLEX/UsrMan/topics/APIs/Cpp/architecture.html. [Accessed 19 04 2017].
- [10] B. MEINDL; M. TEMPL, *Analysis of commercial and free and open source*, Vienna, 2012.
- [11] H. MITTELMANN, *Benchmarks for optimization software*, [Online]. Available: <http://plato.asu.edu/bench.html>. [Accessed 2011].
- [12] NETLIB, *Testcases for real world LP instances*, [Online]. Available: <ftp://ftp.netlib.org/lp/data/index.html>. [Accessed 2012].
- [13] J. L. GEARHART; K. L. ADAIR; R. J. DETRY; J. D. DURFEE; K. A. JONES; N. MARTIN, *Comparison of Open-Source Linear Programming Solvers*, Sandia National Laboratories, Albuquerque, 2013.
- [14] MOXA, *IA240*, [Online]. Available: http://www.moxa.com/product/IA241_IA240.htm. [Accessed 10 05 2017].
- [15] MOXA, *UC-8100-ME-T Series*, [Online]. Available: <http://www.moxa.com/product/UC-8100-ME-T.htm>. [Accessed 10 05 2017].
- [16] HEWLETT PACKARD ENTERPRISE, *QuickSpecs - HPE ProLiant DL360 Generation9 (Gen9)*, 2017.
- [17] A. KAPOULKINE, *pugixml*, [Online]. Available: <http://pugixml.org/>. [Accessed 08 05 2017].
- [18] J. LÖFBERG, *MATLAB 2016 + CPLEX crash*, [Online]. Available: <https://yalmp.github.io/cplexcrash/>. [Accessed 02 06 2017].
- [19] D. THIMSEN, *Costs of Utility Distributed Generators, 1-10 MW*, Epri, Palo Alto, 2003.
- [20] EUROPEAN COMMISSION, *Consumer prices of petroleum products inclusive of duties and taxes*, 2017.
- [21] EUROPEAN ENVIRONMENT AGENCY, *Transport fuel prices and taxes*, [Online]. Available: <https://www.eea.europa.eu/data-and-maps/indicators/fuel-prices-and-taxes/assessment-3>. [Accessed 10 06 2017].

- [22] DIESEL SERVICE AND SUPPLY, *Approximate Diesel Fuel Consumption Chart*, [Online]. Available: http://www.dieselserviceandsupply.com/Diesel_Fuel_Consumption.aspx. [Accessed 10 06 2017].
- [23] IRENA, *Renewable Power Generation Costs in 2014, 2015*.
- [24] K. P. KAIRIES, *Battery storage technology improvements and cost reductions to 2030: A Deep Dive*, IRENA, Düsseldorf, 2017.
- [25] ENERGY STORAGE NEWS, *How to determine meaningful, comparable costs of energy storage*, [Online]. Available: <https://www.energy-storage.news/blogs/how-to-determine-meaningful-comparable-cost-of-energy-storage>. [Accessed 10 06 2017].
- [26] GEMINI ESTORE, *ILOG CPLEX Optimization Studio DE*, [Online]. Available: <http://estore.gemini-systems.com/ibm/software-license/industry-solutions/cplex-optimization-studio/ilog-cplex-optimization-studio-de/>. [Accessed 10 06 2017].



Appendix

In this appendix, the spread sheets for the accumulated cost analysis in chapter 8 are attached. The spreadsheet on this page contains all the necessary assumptions and estimations for the calculations. Rows that are needed for all three cases are marked grey, rows for cases 2 and 3 are marked blue, and rows only used in case 3 are orange.

The three spreadsheets on the next page contain the cost sums for the three cases presented in chapter 8. "Alternative A" corresponds to Case 1, "Alternative B" to Case 2, and "Alternative 3" to Case 3.

Data / Period	0	1	2	3	4	5	6	7	8	9	10
Diesel Maintenance costs (€/unit)	5000.00	5000.00	5000.00	5000.00	5000.00	5000.00	5000.00	5000.00	5000.00	5000.00	5000.00
Number of diesel Generators	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00	9.00
Solver Costs (€)	60000.00	60000.00	60000.00	60000.00	60000.00	60000.00	60000.00	60000.00	60000.00	60000.00	60000.00
Hardware Investment (€)	1000.00										
PV running costs (% of initial investment)	1.50%	1.50%	1.50%	1.50%	1.50%	1.50%	1.50%	1.50%	1.50%	1.50%	1.50%
PV running costs (€)	240000.00	240000.00	240000.00	240000.00	240000.00	240000.00	240000.00	240000.00	240000.00	240000.00	240000.00
Installed PV power (MWp)	8.00										
PV initial investment (€/MW)	2000000.00										
Battery running costs (% of initial investment)	1.50%	1.50%	1.50%	1.50%	1.50%	1.50%	1.50%	1.50%	1.50%	1.50%	1.50%
Battery running costs (€)	10020.00	10020.00	10020.00	10020.00	10020.00	10020.00	10020.00	10020.00	10020.00	10020.00	10020.00
Battery initial investment (€)	668000.00										
Energy demand yearly increase rate	3.00%	3.00%	3.00%	3.00%	3.00%	3.00%	3.00%	3.00%	3.00%	3.00%	3.00%
Demand (kWh)	63379000.00	65280370.00	67238781.10	69255944.53	71333622.87	73473631.56	75677840.50	77948175.72	80288620.99	82695219.62	85176076.21
Diesel (HEL) Price (€)	0.64	0.66	0.67	0.69	0.71	0.72	0.74	0.76	0.78	0.80	0.82
Yearly Diesel Price Increase (€)	2.50%	2.50%	2.50%	2.50%	2.50%	2.50%	2.50%	2.50%	2.50%	2.50%	2.50%
kWh per liter diesel	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
€/kWh	0.16	0.16	0.17	0.17	0.18	0.18	0.19	0.19	0.19	0.20	0.20
Discount rate	5.00%	5.00%	5.00%	5.00%	5.00%	5.00%	5.00%	5.00%	5.00%	5.00%	5.00%



Alternative A / Period	0	1	2	3	4	5	6	7	8	9	10
Energy generated by Diesels (kWh)	63379000.00	65280370.00	67238781.10	69255944.53	71333622.87	73473631.56	75677840.50	77948175.72	80286620.99	82695219.62	85176076.21
Demand (kWh)	63379000.00	65280370.00	67238781.10	69255944.53	71333622.87	73473631.56	75677840.50	77948175.72	80286620.99	82695219.62	85176076.21
Investments (€)											
Fix costs (€)	45000.00	45000.00	45000.00	45000.00	45000.00	45000.00	45000.00	45000.00	45000.00	45000.00	45000.00
Diesel Consumption (l)	15844750.00	16320092.50	16809695.28	17313986.13	17833405.72	18368407.89	18919460.13	19487043.93	20071655.25	20673804.90	21294019.05
Diesel Costs (€)	10140640.00	10705980.68	11302839.10	11932972.38	12598235.59	13300587.23	14042094.97	14824941.76	15651432.26	16523999.61	17445212.59
Total payments (€)	10185640.00	10750980.68	11347839.10	11977972.38	12643235.59	13345587.23	14087094.97	14869941.76	15696432.26	16568999.61	17490212.59
Cash flow (€)	-10185640.00	-10750980.68	-11347839.10	-11977972.38	-12643235.59	-13345587.23	-14087094.97	-14869941.76	-15696432.26	-16568999.61	-17490212.59
Accumulated cash flow (€)		-10185640.00	-20936620.68	-32284459.78	-44262432.17	-56905667.76	-70251254.99	-84338349.95	-99208291.71	-114904723.97	-131473723.59
Actualized cash flow (€)		-10185640.00	-10239029.22	-10292824.58	-10347022.90	-10401621.21	-10456616.80	-10512007.16	-10567789.99	-10623963.20	-10680524.88
Accumulated actualized cash flow (€)		-10185640.00	-20424669.22	-30717493.80	-41064516.70	-51466137.91	-61922754.71	-72434761.87	-83002551.86	-93626515.06	-104307039.94
											-115044513.26
Alternative B / Period	0	1	2	3	4	5	6	7	8	9	10
Energy generated by Diesels (kWh)	51103000.00	53004370.00	54962781.10	56979944.53	59057622.87	61197631.56	63401840.50	65672175.72	68010620.99	70419219.62	72900076.21
Demand (kWh)	63379000.00	65280370.00	67238781.10	69255944.53	71333622.87	73473631.56	75677840.50	77948175.72	80286620.99	82695219.62	85176076.21
Investments (€)	16000000.00										
Fix costs (€)	240000.00	240000.00	240000.00	240000.00	240000.00	240000.00	240000.00	240000.00	240000.00	240000.00	240000.00
Diesel Consumption (l)	12775750.00	13251092.50	13740695.28	14244986.13	14764405.72	15299407.89	15850460.13	16418043.93	17002655.25	17604804.90	18225019.05
Diesel Costs (€)	8176480.00	8692716.68	9239243.50	9817786.89	10430170.47	11078320.47	11764271.54	12490172.75	13258294.03	14071032.92	14930921.73
Total payments (€)	24416480.00	8932716.68	9479243.50	10057786.89	10670170.47	11318320.47	12004271.54	12730172.75	13498294.03	14311032.92	15170921.73
Cash flow (€)	-24416480.00	-8932716.68	-9479243.50	-10057786.89	-10670170.47	-11318320.47	-12004271.54	-12730172.75	-13498294.03	-14311032.92	-15170921.73
Accumulated cash flow (€)		-24416480.00	-33349196.68	-42828440.18	-52886227.08	-63556397.54	-74874718.01	-86878989.56	-99609162.31	-113107456.33	-127418489.25
Actualized cash flow (€)		-24416480.00	-8507349.22	-8597953.29	-8688294.48	-8778375.65	-8868200.25	-8957772.25	-9047096.10	-9136176.72	-9225019.42
Accumulated actualized cash flow (€)		-24416480.00	-32923829.22	-41521782.51	-50210076.99	-58988452.63	-67856652.88	-76814425.13	-85861521.24	-94997697.96	-104222717.38
											-113556347.30
Alternative C / Period	0	1	2	3	4	5	6	7	8	9	10
Energy generated by Diesels (kWh)	48382000.00	50283370.00	52241781.10	54258944.53	56336622.87	58476631.56	60680840.50	62951175.72	65289620.99	67698219.62	70179076.21
Demand (kWh)	63379000.00	65280370.00	67238781.10	69255944.53	71333622.87	73473631.56	75677840.50	77948175.72	80286620.99	82695219.62	85176076.21
Investments (€)	16669000.00										
Fix costs (€)	355020.00	355020.00	355020.00	355020.00	355020.00	355020.00	355020.00	355020.00	355020.00	355020.00	355020.00
Diesel Consumption (l)	12095500.00	12570842.50	13060445.28	13564736.13	14084155.72	14619157.89	15170210.13	15737793.93	16322405.25	16924554.90	17544769.05
Diesel Costs (€)	7741120.00	8246472.68	8781843.40	9348951.79	9949614.49	10585750.59	11259387.41	11972666.52	12727850.14	13527327.94	14379624.12
Total payments (€)	24765140.00	8601492.68	9136863.40	9703971.79	10304634.49	10940770.59	11614407.59	12327686.52	13082870.14	13882347.94	14728644.12
Cash flow (€)	-24765140.00	-8601492.68	-9136863.40	-9703971.79	-10304634.49	-10940770.59	-11614407.59	-12327686.52	-13082870.14	-13882347.94	-14728644.12
Accumulated cash flow (€)		-24765140.00	-33366632.68	-42503496.08	-52207467.87	-62512102.36	-73452872.95	-85067280.37	-973994966.89	-110477837.03	-124360184.97
Actualized cash flow (€)		-24765140.00	-8191897.79	-8287404.45	-8382655.69	-8477648.29	-8572380.04	-8666849.64	-8761056.65	-8855001.48	-8948685.26
Accumulated actualized cash flow (€)		-24765140.00	-32957037.79	-41244442.24	-49627097.93	-58104746.22	-66677126.26	-75343975.89	-84105032.55	-92960034.03	-101908719.29
											-110950829.12