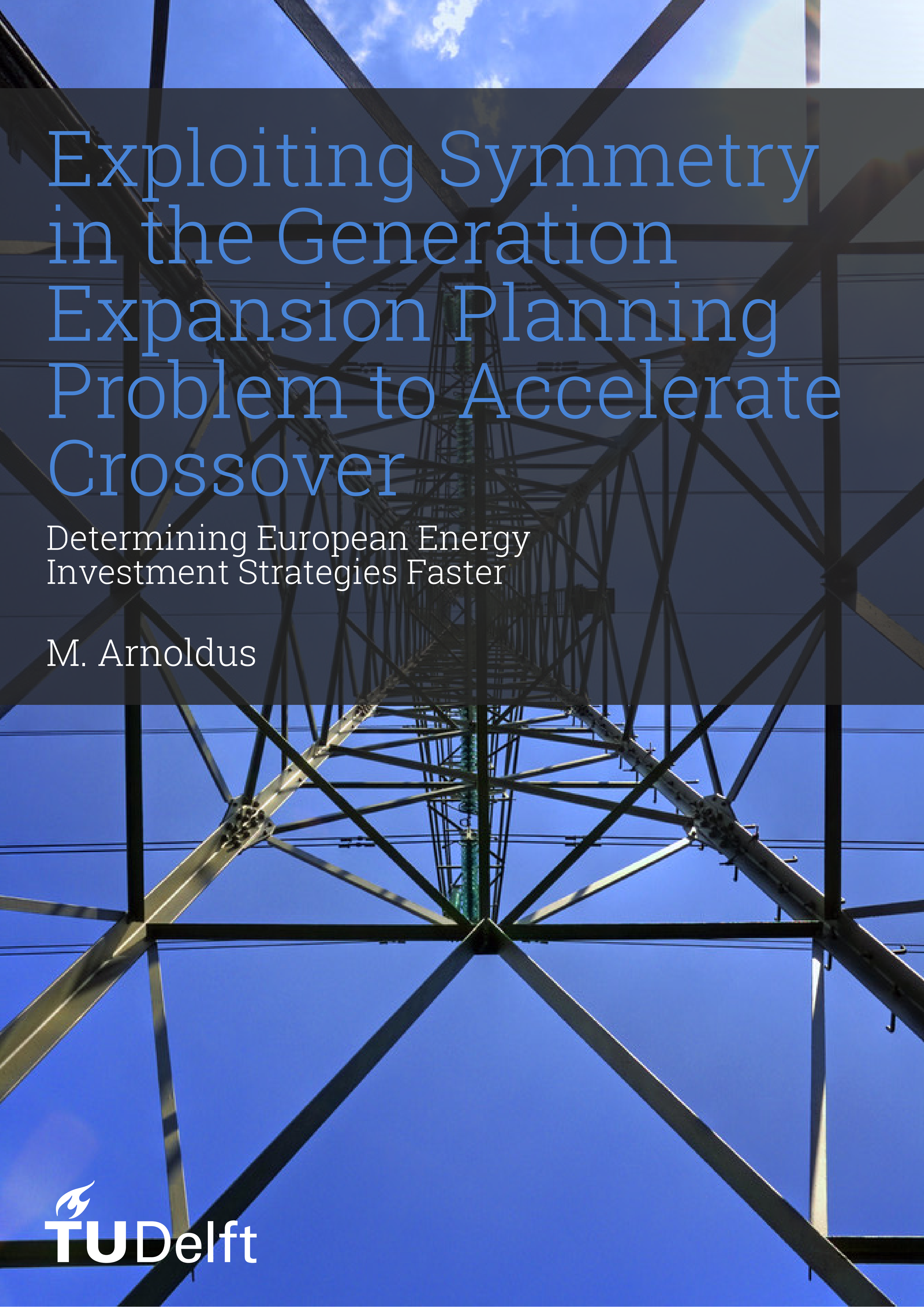


Exploiting Symmetry in the Generation Expansion Planning Problem to Accelerate Crossover

Determining European Energy Investment Strategies Faster

M. Arnoldus

TUDelft

Exploiting Symmetry in the Generation Expansion Planning Problem to Accelerate Crossover

Determining European Energy Investment Strategies Faster

by

M. Arnoldus

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on January 17th 2024 at 13:00.

Student number:	4928091	
Project duration:	April 14, 2023 – January 17, 2024	
Supervised by:	Prof. dr. M.M. de Weerdt,	TU Delft, supervisor
	Dr. G. Neustroev,	TU Delft, daily supervisor
Thesis committee:	Prof. dr. M.M. de Weerdt,	TU Delft, supervisor
	Dr. G. Neustroev,	TU Delft, daily supervisor
	Prof. dr. K. Aardal	TU Delft

TU Delft

Preface

This thesis is the product of a research study I conducted to obtain my master's degree in computer science. Throughout my education I have always enjoyed solving complex puzzles and have developed a growing interest in the fields of mathematics and computer science. Due to my eagerness to delve into challenging problems, I was excited to start this project in one of the interfaces between both research fields. I have enjoyed learning about and deriving creative solutions to a practically-relevant problem during the process of writing this thesis.

I would like to express my gratitude towards everyone whose support has contributed to my work. In particular, I am grateful for the support and guidance of my supervisors, Mathijs de Weerdt and Greg Neustroev. They played a significant role in my project by providing interesting insights and advice in our periodic meetings. For the same reason, I would like to thank Maaike Elgersma, who was involved in helping me find a thesis project and supervising me in the first months.

Furthermore, I want to thank Gérman Morales-España and Diego Tejada Arango from TNO for their efforts in setting up my project. This work would not have been possible without the data and problem formulation they provided to me, and their insightful comments have contributed to the final results of this thesis.

Lastly, I am thankful to my family and friends for supporting me throughout my whole academic career, and in particular the past nine months. Their encouragements have helped me stay motivated and enjoy working on my thesis.

I hope you will enjoy reading this thesis and, for those with a scientific interest in this topic, gain useful insights for future research.

M. Arnoldus
Delft, January 2024

Abstract

Computational efficiency is essential for large-scale mathematical optimisation problems, such as the generation expansion planning problem, to be practically applicable. In linear programming solvers, crossover is frequently a bottleneck when solving optimisation problems. This study aims at determining why crossover is a bottleneck in solving the generation expansion planning problem and deriving a method which obtains an optimal solution faster. Symmetry was found to be disrupting crossover, causing its runtime to significantly increase. However, the preceding barrier method benefit from the symmetry handling of the solver. An alternative approach using problem-specific knowledge to reduce symmetry before serving the model to a solver was proposed. The key to the efficiency of this method is using the information of how symmetries are reduced when retrieving the solution to the original problem. The approach was applied to two types of symmetry we identified in the generation expansion planning problem, leading to a significant speed-up of the crossover phase and total runtime for the algorithm resolving a formulation symmetry, but a longer runtime for the method resolving a more practically available non-formulation symmetry. However, the latter was faster in a couple of cases and has many directions for future improvements. The results of this research, thus, demonstrate that domain knowledge of linear programming problems can be applied to reduce symmetry that state-of-the-art solvers cannot detect and to more efficiently obtain an optimal solution than traditional crossover.

Contents

Nomenclature

Symbols

Symbol	Definition	Unit
$\mathcal{G}$	Set of power generation plants	
$\mathcal{T}$	Set of time steps	
$\mathcal{N}$	Set of nodes in the power market network	
$\mathcal{L}$	Set of energy transmission lines between nodes	
IC_g	Investment cost for power plant g	€/MW
$UCAP_g$	Capacity per invested unit at power plant g	MW
ui_g	Number of units invested in at power plant g	
$PCAP_{n,t}$	Total production capacity of all power plants in node n at time step t	MW
WOP	Period weight (number of hours per time step)	
PC_g	Variable production cost at power plant g	€/MWh
$p_{g,t}$	Energy generation production at plant g at time t	MW
MDC	Costs of missing demand	€/MWh
$md_{n,t}$	Missed demand at node n at time t	MW
$GA_{g,t}$	Generation availability at plant g at time t	
PN_g	Node of power plant g	
NA_l	Node A of transmission line l from A to B	
NB_l	Node B of transmission line l from A to B	
$D_{n,t}$	Energy demand at node n at time t	MW
$ICAP_l$	Maximum transmission import capacity at line l (from B to A)	MW
$ECAP_l$	Maximum transmission export capacity at line l (from A to B)	MW
$f_{l,t}$	Transmission line flow through line l at time t	MW
R	Ramping coefficient	
SUB_g	Set of power plants merged into plant g	
$SUBN_n$	Set of nodes merged into node n	

1 Introduction

Mathematical optimisation is a technology that is applied to many real-world problems. Scheduling, routing and electricity distribution are all types of problems for which an optimal solution can be found using optimisation, or more specifically, using linear programming. As long as such problems remain small enough, linear programming is able to find an optimal solution almost instantly. However, in reality, people desire solutions to problems of extremely large scale, such as finding the route across the whole map or determining the electricity flow in a whole country or continent.

For such large-scale problems it can take hours or even days to find solutions, and at some point it becomes impractical to use linear programming. People have devised many alternatives to linear programming, such as approximation techniques or artificial intelligence methods, but all these techniques do not necessarily end up in the global optimum of the problem. Therefore people dealing with these large-scale problems have to determine whether they favour a quick solution or an optimal solution, and there is always a tradeoff between these two options.

One problem for which this tradeoff exists is the generation expansion planning problem. The objective of this problem is to determine the optimal investment and production decisions for a set of electricity generating technologies in different locations [1]. When this problem is applied to a large scale of, for instance, a whole continent, it can become very complex to solve, and therefore take a lot of time to find the optimal decisions. Especially when one wants to test out many different scenarios, for example, with different costs for technologies or different capacities of electricity transmission lines, it becomes impossible to solve each scenario to optimality. However, a solution close to the optimum might have impractical solution values, like investing extremely small amounts in a particular technology instead of not investing in that technology.

Heuristics, such as rounding investments, can be used to overcome such issues, but ideally linear programming solvers are improved to find the global optimum faster than they currently do. In this thesis we focus on a small part of the solving process for linear programming problems, called the crossover phase. Crossover transforms solutions of interior point methods, which are very close to the global optimum, to the actual optimal solution. Although this seems to be a small task since crossover only needs to transform the solution slightly, it can actually be a very time-consuming process.

Data of one of the biggest companies publishing commercial software for solving linear programming problems, Gurobi, underpins that crossover can require a lot of time. In one of their webinars, Rothberg, CEO at Gurobi, has shown that for 249 models with a runtime of more than 100 seconds, crossover on average takes up almost half of the solving time whenever their interior point method, called the barrier method, is efficiently parallelised [2]. The results of this study can be seen in Figure 1.1: the fractions of runtimes per solving phase are displayed for five scenarios. Each scenario uses a different number of cores to run the barrier method on, and the more cores are used, the faster it is. Already when four cores are used, crossover becomes the new bottleneck of the solving process taking up almost 40% of the total runtime.

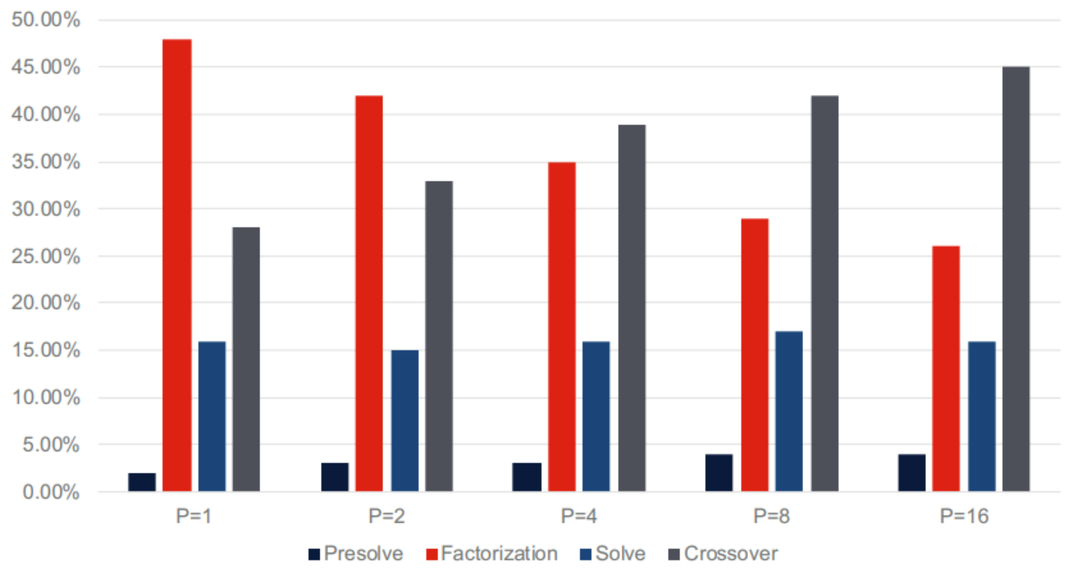


Figure 1.1: Plot showing the fractions of the total runtime for different solving phases when solving a linear programming using Gurobi. The horizontal axis specifies the number of cores on which the barrier method is run in parallel. [2]

The goal of this thesis is twofold. The first aim is to identify reasons for why crossover has a long runtime when solving the generation expansion planning problem. To do this, we use different instances of the problem and investigate which problem characteristics lead to a longer crossover runtime. The second goal is to use this knowledge to determine an alternative to crossover for solving the generation expansion planning problem.

The two aims described above can be split up into four research questions specifying in more detail what we aim to achieve in this thesis. These four questions are as follows:

1. How does crossover perform in terms of runtime and solution quality when solving the generation expansion planning problem?
2. Which problem characteristics of the generation expansion planning problem affect the runtime of crossover and how?
3. What are possible faster alternative methods to crossover that improve upon the quality of an interior point solution?
4. How do the proposed alternatives perform in terms of runtime and optimality of the solution when solving the generation expansion planning problem, compared to crossover and rounding heuristics?

The remainder of this thesis is structured as follows. In the next chapter, we summarise related literature regarding the generation expansion planning problem, crossover and linear programming, and alternative methods to solving optimisation problems. Following this, Chapter 3 discusses the details of the data and mathematical formulation of the generation expansion planning problem used in this thesis. Next, Chapter 4 explains the mathematics behind solving linear programming problems, and crossover in particular. Chapter 5 discusses the performance of crossover on the generation expansion planning problem, and shows that symmetry in the generation expansion planning problem can cause a large crossover runtime, thus concluding the first aim of this thesis.

In Chapter 6 we follow up on this by explaining in detail how linear programming solvers deal with symmetry in problem formulations. After this, we derive which types of symmetry are present in the generation expansion planning problem and propose methods on how to exploit these in Chapter 7. Chapter 8 shows the effects of each type of symmetry on the runtime of crossover and barrier. Furthermore, it shows the results of running the proposed symmetry resolving algorithms. The last chapter of this thesis, Chapter 9, draws conclusions on the results and proposes suggestions for future research.

2
Related Work

To understand why we aim to improve upon crossover in this thesis, it is essential to know its role in solving optimisation problems. Furthermore, it is important to realise why crossover, or mathematical optimisation in general, is relevant for real-world problems such as the generation expansion planning problem.

This chapter summarises relevant literature on both these topics. First, Section 2.1 explains what the generation expansion planning problem entails, why people want to solve it and what techniques other researchers have applied to do so. Second, Section 2.2 reviews the literature on the most used solving technique, mixed-integer linear programming. In particular, we review crossover's part in this technique in order to show why it is an important part of the solving process, to illustrate possible challenges when solving mixed-integer linear programs and to discuss how other researchers have attempted to improve crossover.

2.1. Generation Expansion Planning

As mentioned in the introduction, the generation expansion planning problem determines the optimal type of energy technologies, size, location and time for constructing new (electric) power generators. The aim of the problem is to minimise total costs over a planning horizon, while being subject to certain constraints in the power market [1].

The relevance of this problem has significantly increased over the past decades, but the problem has been around for quite some time. Shortly after the introduction of linear programming, in the 1950s, Massé and Gibrat [3] started researching the problem of determining investments in the electric power industry. Since then, the generation expansion planning problem has evolved and become a popular research topic. The increase in popularity is due to recent and upcoming trends in the use of electricity and renewable energy technologies, such as those described in the 2022 World Energy Outlook by the International Energy Agency [4]. They predict a rise in electricity demand by 5900 to 7000 terawatt-hours by 2030. Some of the reasons for this increase include more electric cars and growing demands in developing economies. Another prediction is that the use of renewable energy sources will rise.

Due to these developments in power markets, the generation expansion planning problem becomes increasingly complex. In a review from 2017 on the integration of renewable energy sources in the generation expansion planning problem, Oree, Hassen, and Fleming [5] explain that the introduction of renewable energy sources in the generation mix has led to more complex models. One reason for this is the intermittent nature of sources as solar and wind energy, leading to more uncertainty in the models. Other modifications to the generation expansion planning problem include objective function and constraint adaptations to account for environmental and sustainability concerns, and energy security.

Another literature review by Koltsaklis and Dagoumas [1] in 2018 argues that there are still a lot of modifications to come. They identify multiple categories of challenges for the generation expansion planning problem that still need to be tackled. These include the integration of electric vehicles, energy storage and the role of policies on the supply of energy. All such developments possibly lead to even more complex problem formulations.

Not only the model formulations have developed over the years, the solution methods evolved alongside them. Many different techniques have been applied to the generation expansion planning problem. Massé and Gibrat [3], the first researchers to publish how they tackled the problem, used linear programming. This is still a frequently used technique to solve generation expansion planning, although researches now often use a mixed-integer linear programming formulation [6], [7]. Sometimes the problem is also formulated as a mixed-integer nonlinear program [8] or as a multi-objective mixed-integer linear program [9] to capture further intricacies of the problem.

Next to mathematical programming, metaheuristic techniques are also applied to solve the generation expansion planning problem. Genetic algorithms are an especially popular tool [10], [11] of this category. A drawback of metaheuristics is that they do not guarantee that a globally optimal solution will be found: this sacrifice allows them to obtain a sufficiently good solution in a reasonable runtime, making these methods suitable for large-scale problems in which the exact optimal solution is not required.

2.2. Mixed-integer Linear Programming

Linear programming refers to a mathematical modeling technique in which a linear objective function is optimised when subjected to a set of linear constraints. To obtain the optimal solution, the values of the so-called *decision variables* $x_j, j = 1, \ldots, n$ are determined. The objective is always to maximise or minimise a linear function, the *objective function*, of these decision variables:

$$\min/\max \, c_1 x_1 + \ldots c_n x_n.$$

Additionally, the decision variables are subject to a set of linear (in)equalities called *constraints*:

$$a_{1,i}x_1 + \cdots + a_{n,i}x_n \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} b_i.$$

An assignment of values to each of the decision variables is called a *solution*, which is *feasible* if it satisfies all of the constraints and *optimal* if it attains the desired minimum or maximum. A linear programming model is referred to as a mixed-integer linear programming model if some of the decision variables are required to be integer [12].

The introduction of linear programming happened in the 1940s. The work of Dantzig [13] on the simplex algorithm is considered the starting point of linear programming research, although some argue that the origins of linear programming are in earlier works. Mixed-integer linear programming developed later, in the 1960s, with the introduction of the branch and bound technique [14]. In the 1980s and 1990s research was more focused on the computational challenges of linear programming. This led to the introduction of barrier methods as a faster alternative to the simplex method for large-scale linear programs [15].

The major difference between the two linear programming techniques, simplex and barrier, is the way in which they traverse the search space, the space of all points satisfying the constraints, to obtain the optimal solution. The simplex algorithm [13] starts by constructing an initial *basic feasible solution*, which is a feasible solution with a minimal number of non-zero decision variables. Visually, a basic feasible solution is an extreme point of the search space. After identifying the initial solution, simplex describes the solution in terms of a *basis* of the non-zero variables and iteratively updates the basis by letting a nonbasic variable which can improve the objective value enter the basis. By doing this, simplex iteratively traverses from a basic feasible solution to another adjacent one until it reaches an optimal basic feasible solution, or in the case of an unbounded problem, an infinite ray on which the objective function is unbounded.

Barrier methods, on the other hand, are a type of interior point method, meaning they start at an arbitrary point in the interior of the search space and iteratively traverse from one interior point to another until they converge to an optimal solution [16]. Barrier methods, in particular, do this by appending a barrier term to the objective function of the linear program. This term approaches positive infinity when any interior point approaches the boundary of the feasible region, in case of a minimisation problem. Therefore, the barrier term prevents the feasible iterates from becoming infeasible by crossing the boundary. However, it discourages iterates to get close to the boundary, which is where the optimal solutions are. Therefore, a weight is used for the barrier term which decreases in its limit to 0 with the number of iterations making the solutions get closer to the optimal solution in every iteration [17].

Figure 2.1 illustrates how the two methods find a solution: the left image shows how simplex traverses the vertices of the search space, while the right images visualises how an interior point method traverses through the interior of the search space.

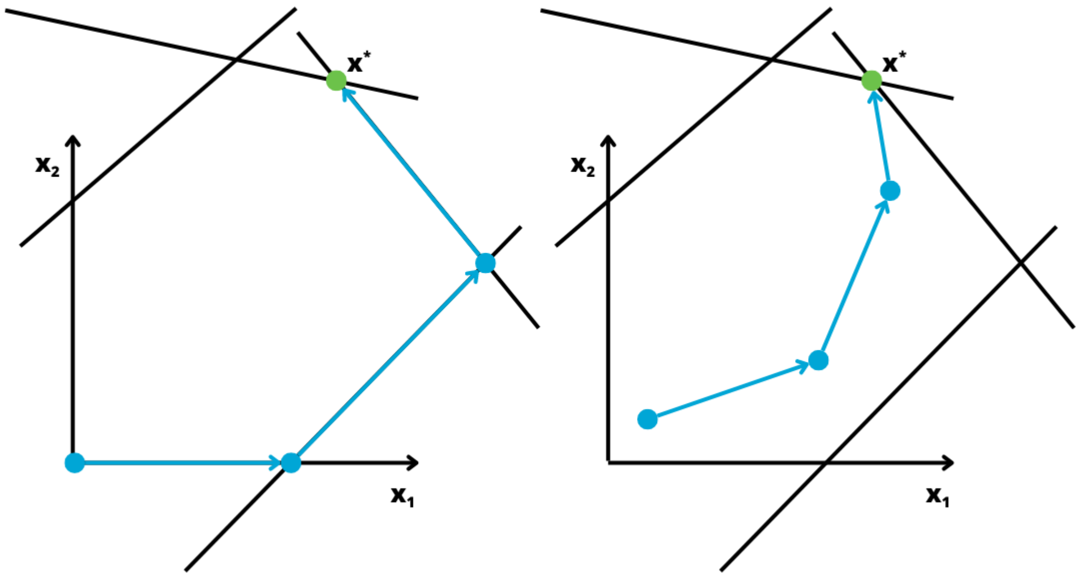


Figure 2.1: Visual representation of how simplex (left) and interior point (right) methods traverse the search space of a linear programming problem to find the optimal solution. Note that this is just an illustration, not a direct translation of how the actual algorithms perform on this example problem.

The simplex method and interior point method are also very relevant to solving mixed-integer linear programming problems. These are typically solved in two stages. For the first stage, all integrality restrictions are removed, resulting in a classic linear programming model called the *relaxed* problem. This is then solved using either the simplex method or an interior point method. In most cases, the resulting solution violates some of the integrality constraints, so it is not feasible to the original mixed-integer linear program. To obtain a solution to the original problem, the branch and bound method is then employed. The idea of this method is to form a tree of sub-problems, starting with the solution to the relaxed problem as the root. Each node branches into at most two nodes. This is done by taking an integer variable of the original problem which has a non-integer value and creating two sub-problems. In one of these problems this variable is bounded above by the largest integer smaller than its value, and in the other it is bounded below by the smallest integer larger than its value. Both sub-problems are optimised again, and this process is repeated until problems return infeasible solutions or problems obtain an optimal solution for which the variables are integer. At termination, one of the leaves of the tree will have an optimal solution to the original mixed-integer linear programming problem [18].

When solving linear or mixed-integer linear programs using interior point methods, crossover is also an important part of the solving process. According to Bixby, Gregory, Lustig, *et al.* [19], interior point methods perform particularly well in the beginning of the solving process, but have trouble making progress when they get closer to an optimal solution. Simplex, on the other hand, struggles progressing away from the initial solution but needs less time when it is already close to an optimal solution. This observation led to the development of a hybrid method, where an interior point method solves the problem up to a certain tolerance and is then followed by a crossover procedure, which consists of two parts. First, a basis is identified for the solution obtained by the interior point method. Then, simplex-like iterations are performed on this basis until an optimal basic feasible solution is reached. Chapter 4 provides more mathematical details on how these two parts work.

As we have shown in Figure 1.1 in the introduction, the crossover procedure can be computationally inefficient. This is acknowledged by Klotz and Newman [20] who state that although crossover typically comprises a small percentage of the run time on most models, it can be time-consuming when initiated on a suboptimal interior point solution with significant difference to an optimal solution.

Although publications on the topic of improving crossover remain scarce, recently the topic has received more interest. In 2020, Schork and Gondzio [21] proposed a new interior point method in which a basis is updated from one interior point iteration to the next. The basis that the algorithm ends up with provides the starting basis for the crossover method.

Another paper from 2021 by Ge, Wang, Xiong, *et al.* [22] also acknowledges that the crossover method often turns out to be the computational bottleneck in practical problems. They propose two new crossover methods. One of those focuses on problems with a network structure, such as the minimum cost flow problem. The other method works on general linear programs. After the first run of the interior point method, they identify the optimal face of the problem. Then, they perturb the problem based on the suboptimal interior point solution. This perturbation is done in a controlled way to ensure that a solution to the perturbed problem is at least as good as the interior point solution, but is instead a feasible extreme point. This method performs well on some of their test problems, but is not consistently faster than crossover methods of traditional solvers.

3 Generation Expansion Planning Problem

Throughout this thesis, we model the generation expansion planning problem as a mixed-integer linear program. Furthermore, we solve this model using input data representing a case study of the European power market. In this chapter we provide a detailed description of the mixed-integer linear program in Section 3.1 and explain the corresponding European case study in Section 3.2.

3.1. Linear-Programming Formulation of the Generation Expansion Planning Problem

The domain of the generation expansion planning formulation used in this thesis consists of nodes, energy sources, power plants, transmission lines and time steps. The set of nodes is denoted by $\mathcal{N}$ and represents the locations at which power plants can be built. Furthermore, each node $n \in \mathcal{N}$ represents a group of consumers demanding energy. Energy can be produced using different energy generation technologies, which form the set $\mathcal{E}$. Each power plant represents a combination between a node and a generation technology and the set of plants is denoted by $\mathcal{G} \subseteq \mathcal{N} \times \mathcal{E}$.

Nodes can also be connected by transmission lines, which allow energy transport between a pair of nodes. The set of transmission lines is denoted by $\mathcal{L} \subseteq \mathcal{N} \times \mathcal{N}$. The last set in the domain of the problem is the set of time steps $\mathcal{T} = \{1, \ldots, T\}$. The larger T is, the longer the planning horizon.

Now that the domain of the problem is defined, we can construct the objective and the constraints of the generation expansion planning problem. We first show the full formulation of the model, and then explain the variables, the objective and each of the constraints and bounds one by one. The full mixed-integer linear programming formulation is given by (3.1).

$$\begin{aligned}
\text{minimize}\quad & \underbrace{\sum_{g\in\mathcal{G}} IC_g\cdot UCAP_g\cdot ui_g}_{\text{investment costs}} + \underbrace{WOP\cdot\left(\sum_{g\in\mathcal{G},t\in\mathcal{T}}(PC_g\cdot p_{g,t}) + \sum_{n\in\mathcal{N},t\in\mathcal{T}}(MDC\cdot md_{n,t})\right)}_{\text{operating costs}} && && \text{(3.1a)}\\
\text{subject to}\quad & && && \\
p_{g,t} &\le GA_{g,t}\cdot UCAP_g\cdot ui_g && \forall g\in\mathcal{G}, t\in\mathcal{T}, && \text{(3.1b)}\\
D_{n,t} &= \sum_{g\in\{g'\in\mathcal{G}|PN_{g'}=n\}} p_{g,t} + \sum_{l\in\{l'\in\mathcal{L}|NB_{l'}=n\}} f_{l,t} - \sum_{l\in\{l'\in\mathcal{L}|NA_{l'}=n\}} f_{l,t} && && \\
&\quad + md_{n,t} && \forall n\in\mathcal{N}, t\in\mathcal{T}, && \text{(3.1c)}\\
f_{l,t} &\ge -ICAP && \forall l\in\mathcal{L}, t\in\mathcal{T}, && \text{(3.1d)}\\
f_{l,t} &\le ECAP && \forall l\in\mathcal{L}, t\in\mathcal{T}, && \text{(3.1e)}\\
p_{g,t}-p_{g,t-1} &\ge -R\cdot UCAP_g\cdot ui_g && \forall g\in\mathcal{G}, t\in\mathcal{T}\setminus\{1\} && \text{(3.1f)}\\
p_{g,t}-p_{g,t-1} &\le R\cdot UCAP_g\cdot ui_g && \forall g\in\mathcal{G}, t\in\mathcal{T}\setminus\{1\} && \text{(3.1g)}\\
p_{g,t} &\ge 0 && \forall g\in\mathcal{G}, t\in\mathcal{T} && \text{(3.1h)}\\
md_{n,t} &\ge 0 && \forall n\in\mathcal{N}, t\in\mathcal{T} && \text{(3.1i)}\\
md_{n,t} &\le D_{n,t} && \forall n\in\mathcal{N}, t\in\mathcal{T} && \text{(3.1j)}\\
ui_g &\ge 0 && \forall g\in\mathcal{G} && \text{(3.1k)}\\
ui_g &\in \mathbb{Z} && \forall g\in\mathcal{G} && \text{(3.1l)}
\end{aligned}$$

In (3.1), the objective (3.1a) is to minimise the total cost, which includes the investment costs and the operating costs. The total investment costs are equal to

$$\sum_{g\in\mathcal{G}} IC_g\cdot UCAP_g\cdot ui_g, \tag{3.2}$$

where IC_g is the investment cost at power plant g in €/MW, $UCAP_g$ is the capacity of one power generation unit at power plant g in MW and ui_g is the number of power generation units built at power plant g. The total operating costs are equal to

$$WOP\cdot\left(\sum_{g\in\mathcal{G},t\in\mathcal{T}}(PC_g\cdot p_{g,t}) + \sum_{n\in\mathcal{N},t\in\mathcal{T}}(MDC\cdot md_{n,t})\right), \tag{3.3}$$

where WOP is the period weight or number of hours per time step t, PC_g is the variable production cost at power plant g in €/MWh, $p_{g,t}$ is the energy generation production at power plant g at time step t in MW, MDC is the cost of the missed demand in €/MW and $md_{n,t}$ is the missed demand at node n at time step t in MW.

The minimisation of the total costs is subject to various types of constraints. The first type ensures that the production never exceeds the capacity of the power plant and is given by (3.1b). In this equation, $GA_{g,t}$ is a load factor representing the generation availability of power plant g at time step t. It accounts for external causes reducing the possible generation, such as the amount of wind or sunlight at a certain time step in a given node; for non-renewable energy sources, $GA_{g,t}$ is typically equal to one.

The second type of constraint ensures the energy balance at each node and is given by (3.1c). In this set of constraints PN_g is the node at which power plant g is located, NA_l and NB_l respectively give the starting node and the ending node of transmission line l, $f_{l,t}$ is the energy transmitted through transmission line l and $D_{n,t}$ is the demand at node n at time step t. These constraints, thus, enforce that the total production at a node, the net energy import at a node and the missed demand at a node add up to the total demand of that node. Therefore, the demand itself is not a hard constraint, but the missed demand $md_{n,t}$ is penalised in the objective function.

Another type of constraints are those ensuring that each transmission line cannot transmit more energy than its capacity. These constraints are given by (3.1d) and (3.1e). Here, $ICAP_l$ and $ECAP_l$ are the maximum import and export capacities of transmission line l in MW. Given a line l from node n_1 to n_2, a negative value for $f_{l,t}$ indicates that energy is transmitted from n_2 to n_1, whereas a positive value means the opposite.

Next, there are ramping constraints. These ensure that power plants do not have massive production changes every time step, which they cannot accommodate in practice. Moreover, depending on the type of energy generation technology used, a power plant cannot go from no production to full capacity production in one time step. In the generation expansion planning problem formulation, these constraints are given by (3.1f) and (3.1g). In those equations R is the ramping rate indicating the maximum difference in production between two time steps as a percentage of the total capacity of that unit, which is assumed equal for each technology.

Furthermore, there are four more bounds that variables need to meet. These are given by (3.1h), (3.1i), (3.1j) and (3.1k). The bounds enforce nonnegativity of production, missed demand, and the number of power generation units, and ensure that the missed demand is always smaller than the actual demand.

Finally, we have the integrality constraint (3.1l), which states that the number of generation units invested in should be integer. In practice, power generators have to be built in full units. However, since the barrier and crossover methods run on the relaxed problem, this constraint is often omitted throughout this thesis.

3.2. Case Study of the European Energy Market

The case study we use in this thesis to run experiments represents the European energy market. In this section, we provide a general overview of this case study by describing the corresponding input data with respect to the problem formulation of the previous section.

The input data of the case study is on country level, meaning that the set of nodes $\mathcal{N}$ consists of European countries or blocs of countries. The following list shows which nodes are present in the data:

- Austria
- Belgium
- Balkan countries
- Baltic countries
- Czech Republic
- Denmark
- Finland
- France
- Germany
- Ireland
- Italy
- Netherlands
- Norway
- Poland
- Portugal
- Slovakia
- Spain
- Sweden
- Switzerland
- United Kingdom

Each of the nodes above is connected to at least one other node by a transmission line. Often, neighbouring countries or blocs are connected to each other. The map in Figure 3.1 shows all connections between the nodes in the list above. Some of these connections might have a different capacity depending on the direction of energy transmission.

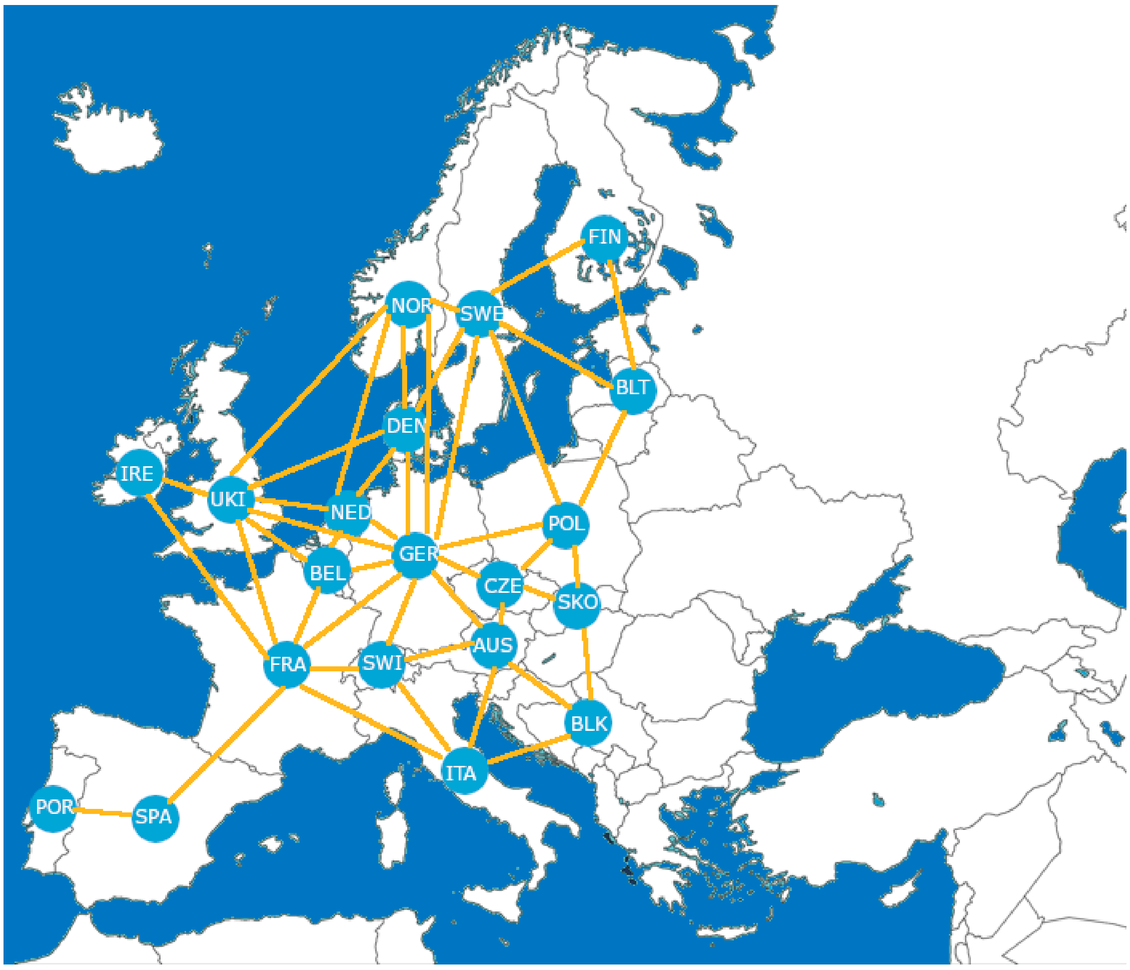


Figure 3.1: Map of European nodes and transmission lines.

Furthermore, each node has a set of power plants in which energy can potentially be generated using a type of generation technology. For this case study, the set of available technologies $\mathcal{E}$ consists of the following:

- Coal
- Gas
- Lignite
- Nuclear
- Oil
- Photovoltaic energy
- Offshore wind
- Onshore wind

The exact values (costs, capacities, etc.) used in this thesis, as well as the code and all results are available online at: https://github.com/marnoldus/accelerate-crossover-using-symmetry.

4 Crossover in Solving Linear Programming Problems

In Chapter 2 we explained that large-scale linear programming problems are typically solved using an interior point method followed by a crossover phase to obtain a basic feasible solution. These two phases are often preceded by a presolve phase in which the problem formulation is reduced to something which can be solved more efficiently. To investigate why crossover takes long on certain problem instances, it is important to understand how the full solving process works in detail.

This chapter provides the mathematical details of this solving process. In Section 4.1, we explain how the solving stages preceding crossover work. These stages include a presolve phase and a barrier method, and they determine the starting point for crossover. Section 4.2 provides a detailed description of how the crossover phase works.

4.1. Preliminary Solving Stages

Since crossover is the last stage in solving a linear programming model, we need to know how the preceding solving stages work, and especially what solution they end up with, to understand the aim and workings of the crossover phase. There are two stages a solver typically undergoes before running crossover, a presolve phase and the interior point method.

Presolving is not required, but most solvers run this before the solving process actually starts. It can be applied before both simplex and interior point methods and has the goal to formulate the linear programming problem in such a way that it is easier to solve. Some rules a well-formulated linear programming model might follow are that it contains as few variables, constraints and nonzeros as possible, that it must be well-scaled and that the constraints are linearly independent [23].

In practice, it is hard to ensure all these rules when formulating a linear programming model. That is the reason why presolving has been used for a long time already. In 1995, Andersen and Andersen [23] summarised presolve methods that were used in solvers of that time. We will explain some of these methods to give an idea of what presolving consists of.

To explain these methods, we consider a linear programming model of the form

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0. \end{aligned} \tag{4.1}$$

Any linear programming problem can be formulated in this way.

The simplest presolving methods consider single variables or single rows of columns of the constraint matrix. A row i can for instance be removed if $a_{ij} = 0 \; \forall j$ and a column is redundant if it only contains zeroes. Furthermore, infeasibility of variables can be detected if its lower bound is bigger than its upper bound. Also, variables can be fixed if their bounds are equal or they are in a singleton row, meaning that only $a_{ik} \neq 0$ and the rest of the factors are equal to 0. In that case x_k can be fixed to b_i/a_{ik}.

More complex presolving methods consider multiple rows or columns at a time. For example, such methods detect linear dependent rows or columns of the constraint matrix A to remove redundancy. Especially if one row/column is a scalar multiple of another row/column, one of these can be removed. Other methods detect if certain constraints can fix all their variables or tighten the bounds on their variables.

Current solvers still apply these presolve methods, but have extended their approaches. A study from 2016 by Achterberg, Bixby, Gu, *et al.* [24] discusses all presolve reductions available in the commercial solver Gurobi version 6.5. Most of the reductions that were not available some time ago consider more complex combinations of constraints or variables, or even look at the whole problem. One example of a reduction looking at the full problem is that Gurobi's presolve method aggregates pairs of symmetric variables. To detect this symmetry all constraints involving either of the variables need to be considered.

The presolving process thus serves as a way to simplify the model that the solving algorithm has to solve. Although many methods remove redundancy from the model, some presolve methods aggregate or remove variables. The drawback of the latter is that these variables have to be reassigned a value after solving the simplified model.

After the presolving process, the simplified model, which is still of the form of (4.1), is served to the solving algorithm used. Here, we discuss how the primal-dual logarithmic barrier algorithm works, which is a type of interior point method. This method was originally developed by Kojima, Mizuno, and Yoshise [25].

Their method works simultaneously on the primal linear programming problem of the form of (4.1) and on the corresponding dual problem. It iteratively generates a sequence of pairs of interior feasible solutions to the primal and the dual. The duality gap of these pairs of solutions, which is the difference between the objectives of the primal and the dual, converges to zero.

The way in which the algorihtm of Kojima, Mizuno, and Yoshise [25] generates each of the pairs of solutions is based on the logarithmic barrier method introduced by Frisch [26] and first applied to a linear program by Gill, Murray, Saunders, *et al.* [27]. Their method transforms a model of the form of (4.1) into

$$\begin{aligned} \text{minimize} \quad & c^\top x - \mu \sum_{j=1}^{n} log x_j \\ \text{subject to} \quad & Ax = b. \end{aligned} \tag{4.2}$$

In the above equation, μ is called the *barrier parameter* and is larger than or equal to 0.

The algorithm for finding an optimal interior point solution works as follows. It starts with choosing an initial interior point x^0 and the initial barrier parameter μ^0. In every iteration, the subproblem (4.2) is solved to obtain a new value for x^{i+1}. Then, the barrier parameter is updated so that $\mu^{i+1} \leq \mu^i$. In the end, as μ^i converges to zero, the value of x^i converges to the optimal solution of (4.1).

Kojima, Mizuno, and Yoshise [25] apply a similar approach to the dual problem to obtain both the optimal primal and dual solutions.

Note that the barrier method converges to an optimal solution of (4.1), not necessarily a basic feasible solution. This means that if there are multiple optimal solutions, it might end up close to any of them instead of in a vertex as simplex does. Moreover, solvers typically use a convergence parameter to determine when the solution of the barrier method is close enough to the optimal solution. As discussed in Section 2.2, crossover takes over from there to obtain an optimal basic feasible solution.

4.2. Crossover

Given an interior point solution to a linear programming problem of the form of (4.1), crossover continues the solving process to obtain an optimal basic feasible solution. As discussed in Section 2.2, crossover consists of two phases: basis identification and a variable push phase. In this section we describe these in Subsections 4.2.1 and 4.2.2, respectively.

4.2.1. Basis Identification

Since crossover performs iterations that are similar to how simplex works, crossover also maintains a basis for its solutions. However, the interior point solution that crossover starts with does not come with a corresponding basis. Therefore, the first phase of crossover is identifying a basis for the interior point solution.

Identifying a basis for an interior point solution can be done in several ways. In this subsection we discuss one method, which is applied in the solver Gurobi as discussed by Maes, Gu, Rothberg, *et al.* [28]. To explain their method, we consider to start with an interior point solution to a problem of the form of (4.1).

Ideally, the initial basis satisfies three conditions. The first condition is that the basis matrix is sparse. The sparsity of a matrix, which is defined as the number of zero entries in the matrix, can influence the computational complexity as denser matrices make matrix multiplications more complex. Next to sparsity, it is important to have a well-conditioned basis. For a well-conditioned basis, a small change in the right-hand side of the problem b does not cause a large change in the variables x. The benefits of a well-conditioned basis are that in the push phase of crossover the solution will not diverge arbitrarily. The last condition is that the basis is close to optimal. This will require a smaller gap to be overcome in the push phase of crossover.

The basis identification method starts with selecting a set of candidate columns

$$C = A(;, candidates),$$

where C consists of all columns whose corresponding variable have small reduced costs. The reason for choosing columns with variables with small reduced costs is to obtain a well-conditioned basis.

The problem of finding a basis is now equivalent to finding m independent columns of C, where m is the number of rows of A. This is done by performing a LU factorisation of $C^\top$. The method of Maes, Gu, Rothberg, *et al.* [28] performs a left-looking LU factorisation, meaning that the following equation is factorised:

$$LU = PC^\top Q, \tag{4.3}$$

where L is a lower triangular matrix and U is an upper triangular matrix. In the equation above, P is a permutation matrix, which when left-multiplied to A reorders the rows of $C^\top$, whereas Q is a permutation matrix which reorders the columns of $C^\top$.

The LU factorisation is used to more efficiently find m independent rows of $C^\top$. These rows are equivalent to columns of C, and therefore equivalent to columns of A. The columns of A which we end up correspond to the variables that are in the initial basis.

4.2.2. Push and Exchange

The initial basis obtained from the procedure described in the previous subsection is not necessarily an optimal basis. To get to an optimal basic feasible solution, commercial solvers like Gurobi and CPLEX use simplex-like push iterations [28], [29]. Their procedure is similar to the method Bixby and Saltzman [30] proposed as a modification of Megiddo's strongly polynomial algorithm for finding an optimal basic solution from any complementary pair of primal and dual feasible solutions [31]. In this subsection we explain the details of the method by Bixby and Saltzman.

Their algorithm starts with any non-optimal basis B for the given linear programming problem, such that $A^*x = Bx_B + Sx_S = b$ and consists of two phases: the primal and the dual phase. In both phases the goal is to reduce the number of primal and dual superbasic values, respectively. A *superbasic* variable is a variable which is not in the current basis and also not at either of its bounds. As both phases work similarly we only discuss the workings of the primal phase. Furthermore, the primal and dual phases can be run in either order, and commercial solvers usually provide parameters for this order.

In the primal phase the algorithm iterates over all superbasic variables $x_j \in x_S$. For each superbasic variable x_j it first tries to push it to its upper or lower bound, by adjusting x_B. If it manages to do so before any of the variables in the basis reaches a bound, x_j is no longer superbasic and the algorithm continues. In case a basic variable reaches a bound before this has happened, x_j is exchanged with that variable and becomes a basic variable. The variable moved out of the basis is now at its bound and is, therefore, also not superbasic. This process is continued until none of the variables are superbasic anymore. Afterwards, the same idea is applied to the dual of the problem in the dual phase. After both the primal and the dual push phases, the algorithm has obtained an optimal basic feasible solution to the linear programming problem.

5 Performance of Crossover when Solving the Generation Expansion Planning Problem

One of the aims of this thesis is to identify causes for crossover to take significantly longer than other solving phases, when solving the generation expansion planning problem. To identify these causes, we run several problem instances with different characteristics. This chapter evaluates the performance of crossover on each of these instances and identifies reasons why crossover does or does not perform well.

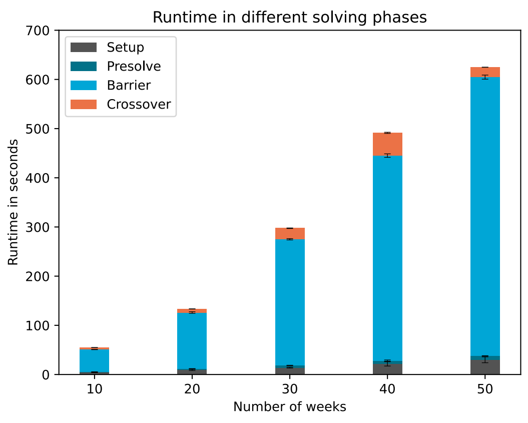
The first section of this chapter runs the baseline problem and discusses the effect of coupling constraints. Section 5.2 introduces a simplified problem instance which highlights the runtime performance issues of crossover. All experiments in this chapter and upcoming chapters are run using optimisation solver Gurobi 10.0.1 [32], unless stated otherwise, on a computer with an AMD Ryzen 7 processor with Radeon Graphics running at 1.80 GHz using 16 GB of RAM and 8 physical cores, running Microsoft Windows 11.

5.1. Crossover Performance on the Baseline Problem

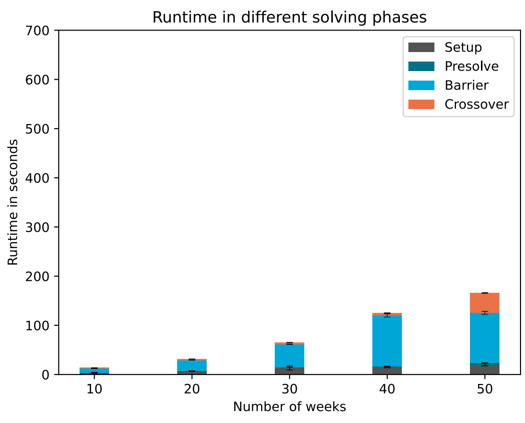
The baseline problem instance for this thesis is the case study described in Section 3.2. For the experiments, we solve the relaxed problem, i.e. (3.1) without the integrality constraints 3.1k, as we are not interested in the performance of branch and bound.

Figure 5.1a shows the runtime of solving the baseline problem instance, separating the runtimes for each of the solving phases. The setup phase consists of supplying the variables, constraints and objective value to the solver. The other three phases are the actual solving phases as described in Chapter 4. For each of the solving phases, the displayed runtime is the mean over five runs, and the error bars correspond to the standard deviation of the five results. The results in Figure 5.1a also show the dependence of the runtime on the time horizon of the problem instance. The horizontal axis displays the number of weeks on which the model is run, ranging from ten to fifty production weeks. The trend in the plot is that the runtime of crossover increases for a longer planning horizon, but so is the case for the barrier method. This trend is also expected, as a longer planning horizon leads to more variables and constraints. Another interesting result is given by the fact that the runtime of the barrier method is still the bottleneck for each of the planning horizons.

Figure 5.1b displays the results of the same problem instance, but without the ramping constraints (3.1f) and (3.1g). The trend is similar to that in Figure 5.1a, only with all runtimes being significantly shorter. Therefore, the drastic increase in constraints due to ramping, which couples each subsequent pair of production values, only influences the total runtime and not specifically that of crossover.



(a) Baseline problem.



(b) Baseline problem without ramping constraints.

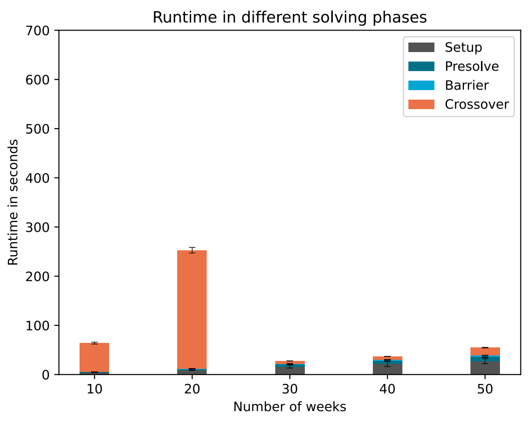
Figure 5.1: Runtime of different solving phases for the baseline problem with production horizons of ten to fifty weeks.

5.2. Crossover Performance on the Symmetric Generation Expansion Planning Problem

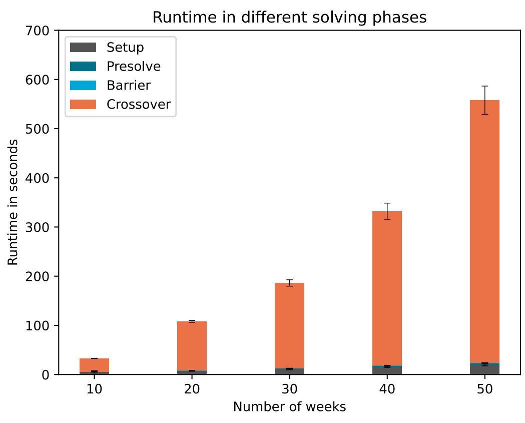
In the previous section we showed problem instances for which the majority of the runtime was not spent in the crossover phase. This section derives an instance for which crossover is the bottleneck and identifies a probable reason for crossover being the bottleneck.

As discussed in Section 4.1, the barrier method converges to any optimal solution in the search space. Therefore, a larger or higher-dimensional set of optimal solutions might be a reason for crossover to take longer, as the starting point of crossover can then be further away from an optimal vertex.

To test the effect of having a higher number of optimal solutions, we propose two simplifications to the problem instance used in the previous section. The first modification is that instead of using actual yearly demand data, we set the demand of each country to the same value of 4000 MW. The second simplification is that the generation availability profile of each renewable power plant is set to a constant availability of 30% of the total availability, i.e. $GA_{g,t} = 0.3$ for every technology g and time step t. Since the investment and variable costs in the original data were already equal for the same technologies in different countries, these modifications cause the problem to have multiple optimal solutions as countries are now almost identical.



(a) Symmetric problem.



(b) Symmetric problem without ramping constraints.

Figure 5.2: Runtime of different solving phases for the symmetric problem with production horizons of ten to fifty weeks.

We run the exact same experiments as in Figure 5.1 using this simplified problem instance. Figure 5.2 displays the results.

In Figure 5.2b, the scenario without ramping constraints, we clearly see an increasing trend in the runtime where almost all the time is spent in the crossover phase. When we compare this to the results for the original problem instance (see Figure 5.1b), we see that the total runtime is significantly larger. This is counter-intuitive because the new problem instance appears simpler due to the larger amount of equal constants.

The results in Figure 5.2a, where ramping constraints are applied, show a different trend. The results for ten and twenty weeks show a similar pattern to that of Figure 5.2b, but starting from thirty weeks the runtime drastically decreases. However, even in these cases, the runtime of crossover is longer than that of the barrier method.

Another interesting observation can be obtained from the logs that Gurobi produces when solving this model. Listing 5.1 shows the part of the logs referring to crossover when solving the simplified problem without ramping constraints on ten weeks of input data.

Listing 5.1: Gurobi solver logs when solving the simplified baseline problem with a planning horizon of 10 weeks.

```
Crossover log...

       1 DPushes remaining with DInf 0.0000000e+00                 1s

      17 PPushes remaining with PInf 0.0000000e+00                 1s
       0 PPushes remaining with PInf 0.0000000e+00                 1s

  Push phase complete: Pinf 0.0000000e+00, Dinf 0.0000000e+00      1s

Iteration    Objective       Primal Inf.    Dual Inf.      Time
      20    6.4134400e+06   0.000000e+00   0.000000e+00      1s

Use crossover to convert LP symmetric solution to basic solution...
Crossover log...

  129578 DPushes remaining with DInf 0.0000000e+00                 5s
   70808 DPushes remaining with DInf 0.0000000e+00                10s
   49792 DPushes remaining with DInf 0.0000000e+00                15s
   24864 DPushes remaining with DInf 0.0000000e+00                20s
    8353 DPushes remaining with DInf 0.0000000e+00                25s
       0 DPushes remaining with DInf 0.0000000e+00                28s

    5055 PPushes remaining with PInf 0.0000000e+00                28s
       0 PPushes remaining with PInf 0.0000000e+00                28s

  Push phase complete: Pinf 0.0000000e+00, Dinf 2.1353757e-09     28s

Iteration    Objective       Primal Inf.    Dual Inf.      Time
  130746    6.4134400e+06   0.000000e+00   0.000000e+00     28s

Solved in 130746 iterations and 28.09 seconds (64.82 work units)
```

The interesting part of the logs above is the fact that crossover now consists of two phases, which was not the case for the tests in the previous section. Moreover, the new second phase is the phase in which almost all the time is spent. This second phase is only necessary for symmetric linear programming problems. For symmetric problems, Gurobi reduces the problem with respect to its symmetries in the presolve phase, so after the first crossover phase a solution to the reduced model is obtained. The second crossover phase is required to unfold the reduced solution to the solution to the original model [33].

To ensure that the runtime issues of crossover are caused by the solving method instead of implementation details, we compare Gurobi's results with the runtime of CPLEX [34], another commercial optimisation solver. Table 5.1 shows the runtimes of Gurobi and CPLEX 22.1.1.0 on the symmetric problem without ramping constraints with a production horizon of fifty weeks. The results are averages over five runs and the standard deviation is given between brackets.

In Table 5.1, we observe that CPLEX is more efficient than Gurobi on this particular problem. However,

Table 5.1: Comparison of average runtime in seconds per solving phase for Gurobi and CPLEX on the symmetric problem instance without ramping and with a planning horizon of fifty weeks. Results are given in seconds and the standard deviation over five runs is given in between brackets.

	Solving phase			
	Setup	Presolve	Barrier	Crossover
Gurobi	19.61 (1.89)	2.89 (0.16)	1.54 (0.07)	533.81 (28.74)
CPLEX	18.67 (1.85)	1.97 (0.98)	0.10 (0.01)	260.46 (10.89)

also for CPLEX, the runtime of crossover is significantly longer than of the other solving phases. In both Gurobi and CPLEX, the crossover phase takes up more than 99% (99.18% and 99.21%, respectively) of the actual solving time, excluding the setup phase.

Apparently, by simplifying the problem such that is has multiple optimal solutions, we have introduced symmetries which solvers are able to detect and deal with. However, the way in which they deal with symmetries seems to harm the runtime of crossover. To investigate this further, we run an experiment with the exact same problem instance disabling Gurobi's symmetry detection. The results of this without ramping constraints are provided in Figure 5.3.

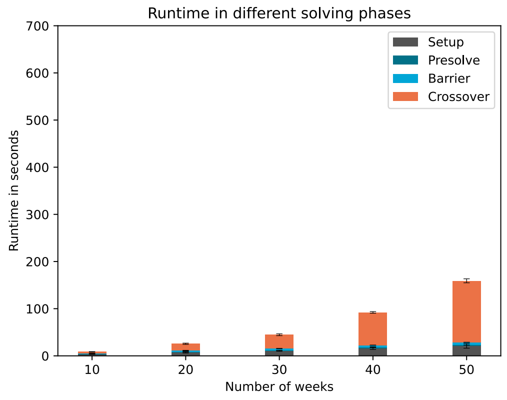


Figure 5.3: Runtime of different solving phases for the symmetric problem with production horizons of ten to fifty weeks. The symmetric problem instance is used and symmetry detection is disabled.

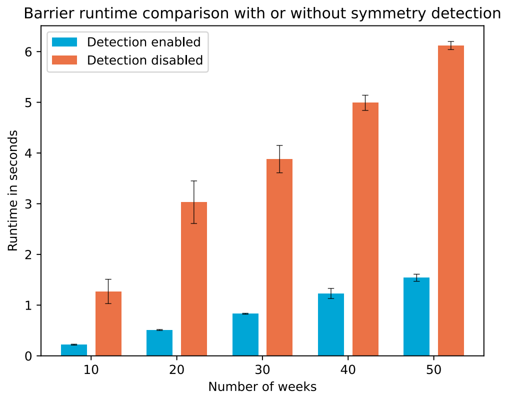


Figure 5.4: Runtime comparison of the barrier method whenever symmetry detection is enabled or disabled. The problem instance used is the symmetric problem and the results are shown for ten to fifty weeks.

Compared to Figure 5.2b, the runtime in Figure 5.3 is significantly smaller, which is caused by a large reduction in the runtime of crossover. It is interesting that for this specific problem instance detecting and handling symmetries is clearly ineffective as its goal is to improve computational efficiency.

Another interesting fact is shown in Figure 5.4. Here we observe that, although the runtime of crossover is much smaller when not detecting symmetry, the runtime of barrier is significantly longer when not detecting symmetry. For this specific problem instance the runtime of barrier is much smaller than that of crossover, so it is less relevant, but it demonstrates that symmetry detection and handling benefits the barrier method.

The results of this section suggest that applying crossover to unfold solutions of symmetric problems is inefficient, while symmetry reductions benefit the runtime of the barrier method. Therefore, ideally, an optimisation solver should resolve symmetry in a manner in which barrier benefits from the reduction without harming the runtime of crossover. In the next chapter, we explain in more detail what symmetry in linear programming entails and how solvers currently deal with symmetry.

6 Symmetry in Linear Programming

Symmetry is frequently present in formulations of linear programming problems. A symmetry appears whenever different parts of the search space can be mapped to each other without changing the structure of the problem. Thus, having symmetry in a linear programming problem enlarges the search space with redundant information. Exploiting symmetry is, therefore, essential for reducing the runtime of linear programming solvers. This chapter illustrates how solvers detect (Section 6.2) and resolve (Section 6.3) symmetry. To do this, required mathematical definitions regarding symmetries and algebraic group theory are introduced in Section 6.1.

6.1. Preliminaries

To illustrate definitions regarding the symmetry of linear programs we consider a linear program of the form:

$$\begin{aligned} \text{maximize} \quad & c^\top x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0, \end{aligned} \tag{6.1}$$

where $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. The feasible region of (6.1) can be described by the polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$.

A *symmetry* of (6.1) is defined as a bijective map $f : \mathbb{R}^n \longrightarrow \mathbb{R}^n$, such that for all $x \in \mathbb{R}^n, x \in P$ if and only if $x \in f(P)$ and $c^\top f(x) = c^\top x$ for every $x \in P$. Feasible solutions of (6.1) are thus mapped to feasible solutions with the same objective value. One problem with this definition of symmetry is that it is based on the feasible region P, which makes it hard to handle efficiently when solving the linear programming problem. Therefore, in practice, one often only considers permutations of variables that leave the description $Ax \leq b$ and the objective function c invariant. Such permutations are referred to as formulation symmetry by Pfetsch and Rehn [35].

For the definition of a formulation symmetry, some more notions regarding algebraic group theory are required. All these notions and definitions, except Definition 6.1.1 and the example given by (6.2), are introduced according to the work of Margot [36]. Let Π^n be the set of all permutations of the set $\{1, \dots, n\}$. A permutation $\pi \in \Pi^n$ is represented by an n-vector, where π_i is the image of i under π. The *composition* of two permutations $\pi^1, \pi^2 \in \Pi^n$, written as $\pi^1 \cdot \pi^2$, is defined as the permutation $h = \pi^1(\pi^2)$. Now, a subset $G \subseteq \Pi$ with the composition operator is a *group* if it contains the identity permutation I and satisfies the following two properties:

1. For any $g^1, g^2 \in G$, we have $g^1 \cdot g^2 \in G$,
2. For any permutation $g \in G$, there exists an inverse permutation $g^{-1} \in G$ such that $g \cdot g^{-1} = g^{-1} \cdot g = I$.

The number of permutations in a group G, denoted by $|G|$, is the order of the group.

Now the definition of a formulation symmetry, according to Pfetsch and Rehn [35], is as follows.

Definition 6.1.1. A permutation $\pi \in \Pi^n$ of variables is a *formulation symmetry* of (6.1) if there exists a permutation $\sigma \in \Pi^m$ such that

- $\pi(c) = c$,
- $\sigma(b) = b$ and
- $A_{\sigma(i),\pi(j)} = A_{ij}$.

The set of all formulation symmetries of the variables in (6.1) is called the *symmetry group* of (6.1), and be written as

$$G^{LP} = \{\pi \in \Pi^n : \pi(c) = c \text{ and } \exists \sigma \in \Pi^m \text{ with } \sigma(b) = b, A_{\sigma(i),\pi(j)} = A_{ij}\}.$$

Moreover, the *orbit* of any $v \in \mathbb{R}^n$ under G^{LP} is

$$orb(v, G^{LP}) = \{w \in \mathbb{R}^n : w = g(v) \text{ for some } g \in G\}.$$

The orbit of v under G^{LP} thus consists of all elements in the domain that can be obtained by performing a permutation from the symmetry group on v, which is equivalent to all elements sharing a symmetry with v.

To illustrate the difference between formulation symmetries and all symmetries, let us look at the following simple linear programming example:

$$\begin{aligned} \text{maximize} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq 3 \\ & x_1 \geq 0 \\ & x_2 \geq 0. \end{aligned} \tag{6.2}$$

In this example, it is clear that there exists a formulation symmetry, because we can replace x_1 with x_2 without changing the objective or any of the constraints. Now, suppose we change the lower bound of x_1 in (6.2) to 1, so we obtain $x_1 \geq 1$. This removes the formulation symmetry because when x_1 and x_2 are now permuted, we cannot permute the constraints anymore to obtain the original problem, since the bounds of both variables are now different. However, when we look at the problem visually, we still observe a symmetry in the search space as can be seen in Figure 6.1.

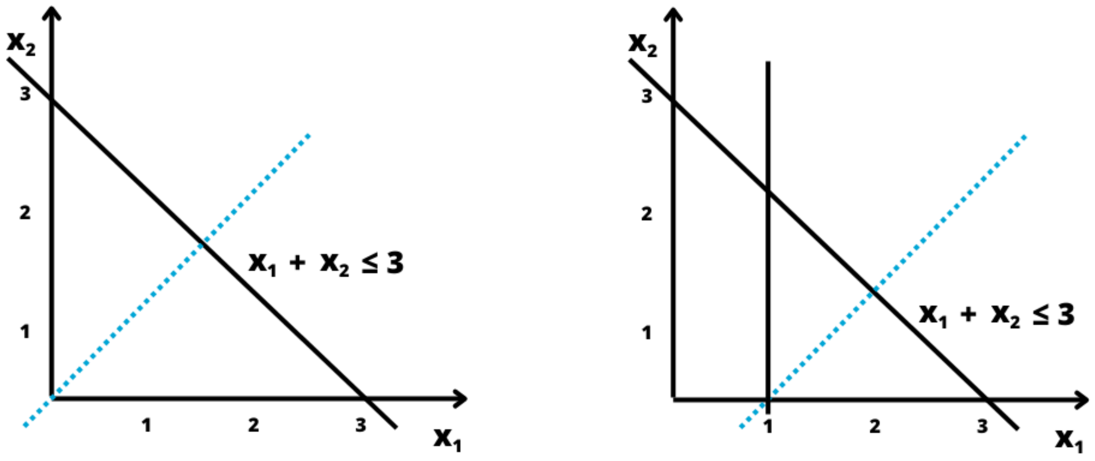


Figure 6.1: Example of symmetry with formulation symmetry (left) and symmetry without formulation symmetry (right).

Figure 6.1 namely shows a visual representation of both situations, with or without the modified constraint $x_1 \geq 1$. In this figure, the blue dotted line shows an axis of symmetry for both situations, indicating that if we map one side of the blue dotted line to the other and vice versa, we obtain a symmetry. The mapping described here is the reflection in the blue dotted line. It is obvious that the mapping preserves the feasible space. The objective value is also constant through this mapping since the blue dotted line is parallel to the optimisation direction. So, even though the second situation does not have a formulation symmetry, the figure shows that there is still symmetry in this problem.

6.2. Symmetry Detection

The first part of how solvers deal with symmetries of a linear programming problem is detecting them. It is important to note that this process only detects formulation symmetries. According to both Margot [36] and Pfetsch and Rehn [35] computing formulation symmetries of a linear programming problem is usually reduced to determining graph automorphisms. A *graph automorphism* is a mapping $f : V \to V$ from vertices of a graph $G = (V, E)$ back to vertices of G such that for each edge $(f(u), f(v))$ in the resulting graph there exists an edge (u, v) in G and vice versa. The reason for detecting symmetries with graph automorphisms is that efficient software tools are available for computing graph automorphisms and a linear programming problem can easily be modelled as a graph. Examples of these software tools are nauty [37], saucy [38] and bliss [39].

To use software tools for computing graph automorphisms, one needs to transform the linear program to a graph. We explain this transformation according to the explanation of Pfetsch and Rehn [35]. In our explanation we use problem (6.1) as our reference. Pfetsch and Rehn explain that a linear program can be modelled as a bipartite graph $G = (V \cup V', E)$ with vertex and edge colours. Here, V consists of the vertices v_j for each x_j in (6.1). These vertices are coloured according to their respective objective coefficient c_j. The second set of vertices, V', contains a vertex for each constraint in $Ax \leq b$, where the colour of the vertex is determined by the right-hand side coefficient b_i. Finally, there is an edge $\{v'_i, v_j\} \in E$ if $A_{ij} \neq 0$, i.e. the variable x_j has a nonzero factor in constraint i. The colour of this edge is according to the coefficient A_{ij}.

Figure 6.2 below shows an example of how a linear programming problem can be transformed to a graph as described above. The procedure above transforms the linear program into the "edge-colored" graph shown in the figure. As can be seen, there are five vertices, three for the three constraints and two for the two variables, as well as four edges, since there are four nonzero elements in the matrix A. Moreover, these edges have two colours because the only nonzero elements in A either have a value of one or two. Furthermore, the vertices have three colours, two for the two different values of the constraints in b, and the two variable nodes have the same colour because their weight in the objective is the same.

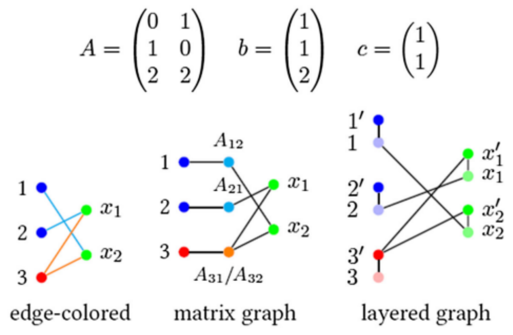

$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 2 & 2 \end{pmatrix}$
$b = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$
$c = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
1
2
3
x_1
x_2
edge-colored
A_{12}
A_{21}
A_{31}/A_{32}
matrix graph
1'
2'
3'
x'_1
x'_2
layered graph

Figure 6.2: Example of a transformation of a linear programming model to a graph used for graph automorphism detection. [35]

Now that we have seen how to transform a linear programming model to a graph, it is important to know how symmetries in the model translate to an automorphism in the graph. This works as follows. A graph automorphism for the bipartite graph G is given by two bijections $\pi : V \to V, \sigma : V' \to V'$ such that $\{\sigma(v'), \pi(v)\} \in E$ for the automorphic graph if and only if $\{v', v\} \in E$ for the original graph. It follows from Definition 6.1.1 that an automorphism in G corresponds to a formulation symmetry in (6.1).

The described transformation from a linear programming problem to a graph still has one problem. State-of-the-art graph automorphism software tools such as the ones mentioned before can namely not deal with coloured edges, but only with coloured nodes. One can easily transform an edge-coloured graph to one without coloured edges by adding coloured vertices. As discussed by Salvagnin [40], an edge $\{v', v\} \in E$ can be replaced by two edges $\{v', w\}, \{w, v\} \in E$, such that w is coloured with the original edge colour. An example of this transformation can again be seen in Figure 6.2. Here the "edge-colored" graph can be transformed into the "matrix graph" when one follows the described procedure.

There is also an alternative method to obtain a graph without coloured edges, which is used in nauty [37]. This method does not depend on bipartiteness and works for a general edge-coloured graph $G = (V, E)$. The result of this method is the "layered graph" in Figure 6.2.

6.3. Symmetry Resolving

After having identified the formulation symmetries in a linear program, solvers can deal with these symmetries. There are several methods that can be used to do this, some of these simply break the symmetry and some really exploit the symmetry to reduce the problem size.

Margot [36] describes a couple of symmetry handling methods. One of these methods is perturbing the problem. This approach introduces small random perturbations in the coefficients of the linear program to destroy the existing symmetries. This might sometimes be helpful, but does not use information from the symmetry and can therefore be counterproductive.

Other methods described by Margot are fixing variables and adding symmetry breaking constraints. The first is a special case of symmetry breaking in which problem information is used to determine variables that can be fixed to a certain value while keeping a subset of the optimal solutions. These variables are then fixed by adding equality constraints. In general, symmetry breaking constraints do not have to deal with single variables, but can also describe relations between different variables to reduce the symmetry in the problem.

The solver SCIP applies a version of adding symmetry breaking inequalities [41]. This works as follows. Given the group of detected symmetries Π, let $A(\Pi) = \{i \in \{1, \dots, n\} : \exists \pi \in \Pi \text{ with } i \neq \pi(i)\}$ be the set of affected variables by symmetries. Select a variable index $\ell \in A(\Pi)$ and compute its orbit $orb(l, \Pi) = \{\pi(\ell) : \pi \in \Pi\}$. ℓ is now called the *leader* of its orbit. Afterwards, all symmetries involving ℓ are removed from the symmetry group and the process is repeated to end up with a group of leaders $\ell_1, \dots, \ell_k$ and their orbits $O_1, \dots, O_k$. By adding the inequalities

$$x_{\ell_i} \geq x_j, \qquad j \in O_i, i \in \{1, \dots, k\},$$

all formulation symmetries in the problem are broken.

Gurobi uses a different approach which reduces the problem using the formulation symmetries in the presolving phase [24]. Their approach works as follows. Again, they start with detecting the formulation symmetry group Π. The symmetry generators $\pi \in \Pi$ can be of two types, *non-overlapping symmetry generators* and *continuous symmetry generators*. A non-overlapping symmetry generator is a generator for which none of the variables permuted by the generator π appears in the same row as one of the other affected variables. A continuous symmetry generator does permute variables which appear in the same row. The problem reduction is first applied to all non-overlapping symmetries and works as follows. Given a symmetry generator π, they aggregate all variables affected by π, i.e. they set $x_j := x_{\pi(j)}$ for all $j \in \{1, \dots, n\}$ with $j \neq \pi(j)$. This is done iteratively to ensure that all variables in the same orbit are aggregated into a single variable. For continuous symmetry generators the exact same approach is used with the exception that it is not applied to integer variables in the case of an integer linear program. The reason for this is that aggregating variables in the same constraint corresponds to computing the arithmetic average of the two, which does not preserve integrality.

An example of the method used by Gurobi is also described by Achterberg, Bixby, Gu, *et al.* [24]. For this example they consider the following linear programming problem.

$$\begin{aligned} \text{maximize} \quad & 4x_1 + 4x_2 + 2x_3 + 2x_4 + 3x_5 + 4x_6 + 5x_7 \\ \text{subject to} \quad & x_1 + 4x_3 + x_5 + 3x_6 + 2x_7 \leq 8 \\ & x_2 + 4x_4 + x_5 + 3x_6 + 2x_7 \leq 8. \end{aligned} \tag{6.3}$$

The symmetry generator π swapping x_1 with x_2 and x_3 with x_4 applies to this problem. If we apply this permutation, we can namely swap the rows to obtain the original problem again. Since neither x_1 and x_2 nor x_3 and x_4 appear in the same row π is a non-overlapping symmetry generator. Aggregating $x_2 := x_1$ and $x_4 := x_3$ now gives us

$$\begin{aligned} \text{maximize} \quad & 8x_1 + 4x_3 + 3x_5 + 4x_6 + 5x_7 \\ \text{subject to} \quad & x_1 + 4x_3 + x_5 + 3x_6 + 2x_7 \leq 8 \\ & x_1 + 4x_3 + x_5 + 3x_6 + 2x_7 \leq 8. \end{aligned} \tag{6.4}$$

Since both constraints are now exactly the same, this can be further reduced to a problem with only one constraint. The symmetry reduction has thus simplified the problem by reducing two variables and one constraint.

Both fixing variables and adding other symmetry breaking constraints suffer from the fact that although the size of the feasible region is decreased, the total number of constraints is increased. This may cause the solving time to not be reduced as much as possible. The method used in Gurobi avoids this by aggregating symmetric variables. However, the description by Achterberg, Bixby, Gu, *et al.* [24] lacks the explanation of how the solution to the original problem is actually retrieved. As we have seen in Listing 5.1 (p. 17) this is done in a second crossover phase, in which each of the variables not at either its upper or lower bound is being pushed towards one of these. The drawback with a lot of symmetry reductions is, therefore, that a lot of variables are being considered in this second crossover phase, which is time-consuming.

7

Symmetry Handling in the Generation Expansion Planning Problem

As explained in previous chapters, symmetry and the way in which it is exploited can affect the runtime of a linear programming solver. In particular, Chapter 5 showed that symmetry reduction can lead to a longer runtime in the crossover method to retrieve a solution to the original problem.

In this chapter we propose a method which reduces symmetry using domain knowledge of the generation expansion planning problem and applies this knowledge to efficiently unfold the solution of the reduced problem to a solution for the original linear program. To achieve this, we first identify two types of symmetry that can be present in the generation expansion planning problem in Section 7.1. The first type involves power plants with equal costs within a node, the second involves symmetric plants in connected nodes. Afterwards, we develop our approach in Section 7.2.

7.1. Symmetries in the Generation Expansion Planning Problem

In this section, we identify and prove the existence of two formulation symmetries in the generation expansion planning problem. Since both of them require assumptions on the input data, we also discuss which assumptions can be relaxed without removing all symmetries. A benefit of using domain knowledge is that we can also identify and resolve the remaining symmetry, even though this cannot be detected from the problem formulation.

The first type of symmetry we discuss occurs whenever two power plants in the same node are almost identical and is described in Section 7.1.1. The second type is a symmetry in which the production of two power plants in two different nodes can be interchanged. This type of symmetry is explained in Section 7.1.2.

7.1.1. Interchangeable Power Plants within a Node

The first symmetry is fairly straightforward. It occurs whenever two power plants in the same node have the same variable and investment costs, unit capacity and generation availability. In that case, the total production costs are unaffected by the decision of employing either of these power plants. One could use one of the plants for producing the total demand of energy, but it could also split the production equally, or even randomly, among both power plants. Each of these divisions result in the same objective value.

In Theorem 7.1.1 we prove that this symmetry is a formulation symmetry according to Definition 6.1.1.

Theorem 7.1.1. Let two power plants $g_1 = (n, e_1)$, $g_2 = (n, e_2)$ in the same country n use different energy sources e_1 and e_2. Suppose the unit capacities ($UCAP_{g_1} = UCAP_{g_2}$), investment costs ($IC_{g_1} = IC_{g_2}$), variable production costs ($PC_{g_1} = PC_{g_2}$) and generation availability ($GA_{g_1,t} = GA_{g_2,t}$ for all $t \in \mathcal{T}$) for both power plants are the same at all time points. Then, the permutation π substituting ui_{g_1} with ui_{g_2} and $p_{g_1,t}$ with $p_{g_2,t}$ for all $t \in \mathcal{T}$ is a formulation symmetry.

Proof. Suppose g_1 and g_2 satisfy the assumptions of the theorem. According to Definition 6.1.1 we need to show three things. First of all, we show that the objective remains unchanged under the permutation π, i.e. $\pi(c) = c$. To do this, we consider the part of the objective consisting only of parameters involving g_1 and g_2. The rest of the objective remains unchanged, and can therefore be disregarded. This gives us the following objective.

$$\begin{aligned} \text{minimize} \quad & IC_{g_1} \cdot UCAP_{g_1} \cdot ui_{g_1} + IC_{g_2} \cdot UCAP_{g_2} \cdot ui_{g_2} + \\ & WOP \cdot \left(\sum_{t\in\mathcal{T}} (PC_{g_1} \cdot p_{g_1,t} + PC_{g_2} \cdot p_{g_2,t}) + \sum_{t\in\mathcal{T}} (MDC \cdot md_{n,t}) \right) \end{aligned} \tag{7.1}$$

Since $IC_{g_1} = IC_{g_2}$ and $UCAP_{g_1} = UCAP_{g_2}$, we can substitute ui_{g_1} and ui_{g_2} without changing the objective function. Next to that, substituting $p_{g_1,t}$ with $p_{g_2,t}$ for all $t \in \mathcal{T}$ also does not change the objective, because $PC_{g_1} = PC_{g_2}$.

Now, we need to find a permutation σ of the constraint rows such that the right-hand side vector b remains unchanged and permuting the left-hand side matrix A using both π and σ returns the original matrix again. To construct σ, let us write all constraints involving at least one of the variables affected by π. All other constraints can stay in their original row, since the column permutations do not change these rows. The affected constraints are the following.

$$\begin{aligned}
p_{g_1,t} &\le GA_{g_1,t} \cdot UCAP_{g_1} \cdot ui_{g_1} && \forall t \in \mathcal{T} && (7.2a)\\
p_{g_2,t} &\le GA_{g_2,t} \cdot UCAP_{g_2} \cdot ui_{g_2} && \forall t \in \mathcal{T} && (7.2b)\\
p_{g_1,t} + p_{g_2,t} + \sum_{l\in\{l'\in\mathcal{L} \mid NB_{l'}=n\}} f_{l,t} - \sum_{l\in\{l'\in\mathcal{L} \mid NA_{l'}=n\}} f_{l,t} + md_{n,t} &= D_{n,t} && \forall t \in \mathcal{T} && (7.2c)\\
p_{g_1,t} - p_{g_1,t-1} &\ge -R \cdot UCAP_{g_1} \cdot ui_{g_1} && \forall t \in \mathcal{T} && (7.2d)\\
p_{g_1,t} - p_{g_1,t-1} &\le R \cdot UCAP_{g_1} \cdot ui_{g_1} && \forall t \in \mathcal{T} && (7.2e)\\
p_{g_2,t} - p_{g_2,t-1} &\ge -R \cdot UCAP_{g_2} \cdot ui_{g_2} && \forall t \in \mathcal{T} && (7.2f)\\
p_{g_2,t} - p_{g_2,t-1} &\le R \cdot UCAP_{g_2} \cdot ui_{g_2} && \forall t \in \mathcal{T} && (7.2g)\\
p_{g_1,t} &\ge 0 && \forall t \in \mathcal{T} && (7.2h)\\
p_{g_2,t} &\ge 0 && \forall t \in \mathcal{T} && (7.2i)
\end{aligned}$$

The permutation σ should substitute multiple pairs of rows with each other. Along with the permutation of the variables, π, this should return the original problem again. First of all, the constraints (7.2a) and (7.2b) should be swapped for each $t \in \mathcal{T}$. Since $GA_{g_1,t} = GA_{g_1,t}$ for all $t \in \mathcal{T}$ and $UCAP_{g_1} = UCAP_{g_2}$, it is clear that permuting these constraints will undo the permutation π of the variables in these two rows. The constraint (7.2c) does not need to be swapped with another row, because the permutation of the production values in this equation does not affect the constraint as they are simply summed. For the ramping constraints, σ has to substitute (7.2d) with (7.2f) and (7.2e) with (7.2g), as these pairs of constraints are exactly the same apart for the values permuted by π. Finally, σ needs to swap the rows (7.2h) and (7.2i). This substitution is trivial as both variables are greater or equal to 0. All these substitutions together form the permutation σ, which can undo the permutation π, thus proving that π generates a formulation symmetry. □

It is important to note that the symmetry described above can of course be extended to more than two power plants. In the case when there are more than two power plants in the same node with the exact same parameters, there also exist multiple permutations π generating a formulation symmetry. With three power plants g_1, g_2 and g_3 for instance, π can permute g_1 to g_2, g_2 to g_3 and g_3 to g_1 or the other way around. Also a subset of two of the three plants also has a formulation symmetry.

Another important note is that for this formulation symmetry to hold, we need to have the generation availability of both power plants to be equal for every time step $t \in \mathcal{T}$. The reason for this is that the investments ui_{g_1} and ui_{g_2} are done beforehand. Therefore, they affect all production values at every time step, which can also be seen by the relation between the production and investment values in constraints (7.2a), (7.2b) and (7.2d) to (7.2g). We can thus not simply swap the production values at one time step if the investment decision does not support either of the amounts produced.

However, in the case that the generation availability is not equal for every time step, there still could be symmetry, although we cannot detect it from the formulation of the problem. A reason for this symmetry to exist is that in an optimal scenario investment in both power plants may be required, for instance, because each of the plants is the only available option at some of the time steps, and therefore needs to be invested in to provide power during those time steps. In that case, it does not matter at individual time steps which one produces the energy as long as they together produce enough energy and each power plant does not violate their capacity and ramping constraints. Therefore, even though two plants with equal costs have different availability profiles, symmetries can still be present in the problem.

In practice, the symmetry described in this subsection is not very likely to occur, especially the formulation symmetry. Different power plants in the same node usually have different technologies, which are unlikely to have the same costs or generation availability profiles. However, from a user perspective there could be reasons for having such a symmetry in the problem. One might for instance have aggregated data from a national level where identical plants existed, or one might be testing multiple scenarios with two plants of which one scenario suggests them having identical costs. Another reason for this symmetry to be present in the data is the absence of real-world data: in this case users often fill in these missing inputs using default values, which are more likely to be the same. So, it is still very useful to identify and resolve this symmetry, especially because it is a relatively simple symmetry which can help in finding and resolving other symmetries.

7.1.2. Interchangeable Power Plants in Connected Nodes

The second type of symmetry we describe is more complex, but more likely to occur in real-world data, than the first. The similarities with the symmetry described in Subsection 7.1.1 are that this symmetry involves two power plants with the same costs, capacities and generation availability profiles. However, in this second symmetry these power plants are located in two different nodes that are connected by transmission lines. The difficulty for this symmetry lies in the fact that now the capacities of the transmission lines and the demands of the different countries influence the production and investments in both plants, and thus the symmetry.

Since this symmetry involves many more variables than the first symmetry, we introduce it step by step. First, we show that for a scenario with strong assumptions, such as infinite transmission line capacity, there exists a formulation symmetry. Afterwards, we remove these assumptions one by one and show how they affect the symmetry and whether we are still dealing with formulation symmetry.

Infinite Transmission Capacity

For the simplest scenario with infinite transmission lines consider the following. Two neighbouring countries each only have one power plant and both plants have the same costs, capacity per unit and generation availability profile. For non-renewable energy sources this happens often in practice, as it is very logical that producing electricity using, for instance, coal is not cheaper in one country than in another. Furthermore, let these two nodes be disconnected from other nodes and have the same demand profile, and assume the bounds on the capacity of the transmission lines ((3.1d) and (3.1e)) are gone, i.e. they have infinite capacity. If all these criteria are met, the production of energy can be done in either of these countries. It namely does not matter if the production is done in one country and partly transported to the other, or the other way around. Both methods have the exact same costs.

For this simplified scenario, we can show that the symmetry described above is a formulation symmetry. To do this, we again need to formulate the problem mathematically, which is done in Theorem 7.1.2. In this theorem we prove a slightly more general version than the scenario above, as we consider that the two connected countries have identical sets of power plants, which can be more than one power plant per country.

Theorem 7.1.2. Let two connected nodes n_1 and n_2 have an identical set of power plants $\{g_{n_1,1}, \dots, g_{n_1,m}\}$ and $\{g_{n_2,1}, \dots, g_{n_2,m}\}$, where each pair of power plants $(g_{n_1,i} g_{n_2,i})$ has identical unit capacities ($UCAP_{g_{n_1,i}} = UCAP_{g_{n_2,i}}$), investment costs ($IC_{g_{n_1,i}} = IC_{g_{n_2,i}}$), variable production costs ($PC_{g_{n_1,i}} = PC_{g_{n_2,i}}$) and generation availability ($GA_{g_{n_1,i},t} = GA_{g_{n_2,i},t}$ for all $t \in \mathcal{T}$). Furthermore, assume both nodes are not connected to other nodes, have the same demand at each time step ($D_{n_1,t} = D_{n_2,t} \forall t \in \mathcal{T}$) and the capacity of the transmission lines between the two nodes is infinite. Then, the permutation π substituting ui_{g_1} with ui_{g_2}, $p_{g_1,t}$ with $p_{g_2,t}$ for all $t \in \mathcal{T}$, $md_{n_1,t}$ with $md_{n_2,t}$ and $f_{l_1,t}$ with $f_{l_2,t}$ for all $t \in \mathcal{T}$, where l_1 goes from n_1 to n_2 and l_2 the other way around, is a formulation symmetry.

Before proving this symmetry, it is important to note that the way in which the decision variables $f_{l,t}$ for flows in the formulation in (3.1) are interpreted can be ambiguous. Based on the formulation, one could namely have two variables $f_{l_a,t}$ and $f_{l_b,t}$ where l_a goes from A to B and l_b from B to A. However, each flow is bounded by both an import capacity and an export capacity in constraints (3.1d) and (3.1e). This suggests that there should only be one variable which is positive if the flow goes from A to B and negative if it flows in the other direction.

Theorem 7.1.2 only holds in the first case, although both problems are equivalent. The reason for this is that in the first case we can swap the variables for the flows, whereas in the second case we cannot since there is only one variable. Since linear programming solvers often use a presolve phase to transform the problem into a simpler equivalent problem, it is likely that if it can detect symmetry for one of the two versions of the problem, it can also do so for the other equivalent version. Therefore, we prove this theorem for the case with two flows between A and B, which, with this transformation of the problem, also proves the symmetry for the other case. The proof for Theorem 7.1.2 now follows below.

Proof. Assume n_1 and n_2 are two nodes connected by an edge $e = (n_1, n_2)$, and suppose that their power plants are identical and satisfy the assumptions made in the theorem. Also, assume that the demand profiles of n_1 and n_2 are equal and that the transmission capacity of e is infinite in both directions. Let π be the permutation as described in the theorem. We now show the three properties of Definition 6.1.1.

First of all, consider the part of the objective consisting of parameters related to n_1 or n_2. The rest is unaffected by π, so remains unchanged. The objective we obtain by doing this is as follows.

$$\begin{aligned} \text{minimize} \quad & \sum_{g_1 \in \{g' \in \mathcal{G} \mid PN_{g'} = n_1\}} IC_{g_1} \cdot UCAP_{g_1} \cdot ui_{g_1} + \sum_{g_2 \in \{g' \in \mathcal{G} \mid PN_{g'} = n_2\}} IC_{g_2} \cdot UCAP_{g_2} \cdot ui_{g_2} + \\ & WOP \cdot \left(\sum_{g_1 \in \{g' \in \mathcal{G} \mid PN_{g'} = n_1\}} \sum_{t \in \mathcal{T}} (PC_{g_2} \cdot p_{g_2,t}) + \sum_{g_2 \in \{g' \in \mathcal{G} \mid PN_{g'} = n_2\}} \sum_{t \in \mathcal{T}} (PC_{g_2} \cdot p_{g_2,t}) + \right. \\ & \left. \sum_{t \in \mathcal{T}} (MDC \cdot md_{n_1,t}) + \sum_{t \in \mathcal{T}} (MDC \cdot md_{n_2,t}) \right) \end{aligned} \tag{7.3}$$

For each g_1 at node n_1 there exists a plant g_2 at n_2 such that the costs are the same ($IC_{g_1} = IC_{g_2}$, $PC_{g_1} = PC_{g_2}$ and $UCAP_{g_1} = UCAP_{g_2}$). Thus, swapping all the ui_{g_1} with all ui_{g_2} does not influence the objective function, nor does swapping the $p_{g_1,t}$ with the $p_{g_2,t}$ for all $t \in \mathcal{T}$. Furthermore, it can be seen that swapping $md_{n_1,t}$ with $md_{n_2,t}$ for all $t \in \mathcal{T}$ does not affect the objective function as both are multiplied by the same missed demand cost MDC. So, the objective remains unchanged under the permutation π.

Now, it remains to find a permutation σ of the constraints such that the right-hand side remains unchanged, and permuting the problem with both π and σ returns the original problem. We write all constraints involving either n_1 or n_2 to show what rows have to be swapped by σ. These constraints are the following.

$$
\begin{aligned}
p_{g_1,t} &\leq GA_{g_1,t} \cdot UCAP_{g_1} \cdot ui_{g_1} && \forall g_1 \in \mathcal{G} | PN_{g_1} = n_1, \forall t \in \mathcal{T} && \text{(7.4a)}\\
p_{g_2,t} &\leq GA_{g_2,t} \cdot UCAP_{g_2} \cdot ui_{g_2} && \forall g_2 \in \mathcal{G} | PN_{g_2} = n_2, \forall t \in \mathcal{T} && \text{(7.4b)}\\
D_{n_1,t} &= \sum_{g_1 \in \{g' \in \mathcal{G} | PN_{g'} = n_1\}} p_{g_1,t} + \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = n_1\}} f_{l,t} && &&\\
&- \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = n_1\}} f_{l,t} + md_{n_1,t} && \forall t \in \mathcal{T} && \text{(7.4c)}\\
D_{n_2,t} &= \sum_{g_2 \in \{g' \in \mathcal{G} | PN_{g'} = n_1} p_{g_2,t} + \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = n_2\}} f_{l,t} && &&\\
&- \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = n_2\}} f_{l,t} + md_{n_2,t} && \forall t \in \mathcal{T} && \text{(7.4d)}\\
p_{g_1,t} - p_{g_1,t-1} &\geq -R \cdot UCAP_{g_1} \cdot ui_{g_1} && \forall g_1 \in \mathcal{G} | PN_{g_1} = n_1, \forall t \in \mathcal{T} && \text{(7.4e)}\\
p_{g_1,t} - p_{g_1,t-1} &\leq R \cdot UCAP_{g_1} \cdot ui_{g_1} && \forall g_1 \in \mathcal{G} | PN_{g_1} = n_1, \forall t \in \mathcal{T} && \text{(7.4f)}\\
p_{g_2,t} - p_{g_2,t-1} &\geq -R \cdot UCAP_{g_2} \cdot ui_{g_2} && \forall g_2 \in \mathcal{G} | PN_{g_2} = n_2, \forall t \in \mathcal{T} && \text{(7.4g)}\\
p_{g_2,t} - p_{g_2,t-1} &\leq R \cdot UCAP_{g_2} \cdot ui_{g_2} && \forall g_2 \in \mathcal{G} | PN_{g_2} = n_2, \forall t \in \mathcal{T} && \text{(7.4h)}\\
p_{g_1,t} &\geq 0 && \forall g_1 \in \mathcal{G} | PN_{g_1} = n_1, \forall t \in \mathcal{T} && \text{(7.4i)}\\
p_{g_2,t} &\geq 0 && \forall g \in \mathcal{G} | PN_{g_2} = n_2, \forall t \in \mathcal{T} && \text{(7.4j)}\\
md_{n_1,t} &\geq 0 && \forall t \in \mathcal{T} && \text{(7.4k)}\\
md_{n_2,t} &\geq 0 && \forall t \in \mathcal{T} && \text{(7.4l)}\\
md_{n_1,t} &\leq D_{n_1,t} && \forall t \in \mathcal{T} && \text{(7.4m)}\\
md_{n_2,t} &\leq D_{n_2,t} && \forall t \in \mathcal{T} && \text{(7.4n)}\\
ui_{g_1} &\geq 0 && \forall g_1 \in \mathcal{G} | PN_{g_1} = n_1, \forall t \in \mathcal{T} && \text{(7.4o)}\\
ui_{g_2} &\geq 0 && \forall g_2 \in \mathcal{G} | PN_{g_2} = n_2, \forall t \in \mathcal{T} && \text{(7.4p)}
\end{aligned}
$$

First of all, let σ swap the constraints in (7.4a) with the constraint in (7.4b). This returns the original constraints after applying π since for each g_1 in node n_1 there exists a g_2 in node n_2 with the same generation availability for each $t \in \mathcal{T}$ ($GA_{g_1,t} = GA_{g_2,t}$ for all $t \in \mathcal{T}$) and capacity ($UCAP_{g_1} = UCAP_{g_2}$). Next, let σ permute the constraints (7.4c) and (7.4d) for all $t \in \mathcal{T}$. Since π swaps all variables on the right-hand side of these constraints and the demands $D_{n_1,t}$ and $D_{n_2,t}$ are equal for all $t \in \mathcal{T}$, swapping these rows gives us back the original constraints. Regarding the ramping constraints, σ has to swap (7.4e) with (7.4g) and (7.4f) with (7.4h) for the same reasons as for the first two constraints. Furthermore, the bounds in (7.4i) have to be substituted with the bounds in (7.4j) by σ, since π substitutes each of the $p_{g_1,t}$ with the $p_{g_2,t}$. The same is the case for (7.4k) and (7.4l), (7.4m) and (7.4n), and (7.4o) and (7.4p). Note that swapping (7.4m) and (7.4n) returns the same right-hand side since the demands are assumed to be equal in every time step $t \in \mathcal{T}$.

Applying the permutation σ, which consists of all swaps as described above, thus returns the original left-hand side of the problem after π has performed its permutation, and σ leaves the right-hand side of the constraints unchanged. Therefore, we have shown that π is a formulation symmetry. □

It is important to note that for a formulation symmetry to exist, n_1 and n_2 should be disconnected from other nodes. Otherwise, Theorem 7.1.2 does not hold, since swapping (7.4c) and (7.4d) then does not return the original problem due to the other flows involved. However, even if n_1 or n_2 is connected to different nodes, there can still be symmetry since the costs of the two power plants are still identical.

The same holds for many of the other assumptions we made in Theorem 7.1.2. It is interesting to identify which of these could be released while keeping symmetry, since releasing these assumptions could lead to more realistic scenarios where symmetry occurs. In the remaining of this subsection, we discuss whether these assumptions are required for a problem instance to be symmetric.

Bounded Transmission

Before, we assumed that the transmission capacity was infinite, which is of course not realistic. The reason we made this assumption was to first show how two power plants in different nodes can be symmetric without adding too many details at once. In this subsection, we release the assumption of infinite transmission capacity.

Without infinite transmission capacity, each transmission line l from node n_1 to n_2 has a maximum import capacity $ICAP_l$ bounding the flow from n_2 to n_1, and a maximum export capacity $ECAP_l$ bounding the flow from n_1 to n_2. As discussed previously, the formulation of the flows determines whether a formulation symmetry exists, and since this formulation is modified in the presolve phase we assume that the symmetry can be found by a solver. Therefore, similarly to in the previous subsection, we assume that the flow from n_1 to n_2 is modelled separately from the flow from n_2 to n_1. Consequently, $ICAP_{(n_1,n_2)}$ means the exact same as $ECAP_{(n_2,n_1)}$.

By doing that, given a line l from n_1 to n_2, the bounds on the flows change from $f_{(n_1,n_2),t} \geq -ICAP_{(n_1,n_2)}$ and $f_{(n_1,n_2),t} \leq ECAP_{(n_1,n_2)}$ for all $t \in \mathcal{T}$ to $f_{(n_1,n_2),t} \leq ECAP_{(n_1,n_2)}$ and $f_{(n_2,n_1),t} \leq ECAP_{(n_2,n_1)}$. Also both flows should now be bounded below by zero, instead of the previous import capacity. Note that this transformation does not change anything to the problem structure, it simply splits each undirected transmission line into two directed transmission lines. So now $f_{(n_2,n_1),t}$ is equal to what previously would be $-f_{(n_1,n_2),t}$.

Using this transformation and the proof of Theorem 7.1.2, one can easily show that even with transmission lines with limited capacity symmetry exists. The theorem remains exactly the same, apart for the assumption regarding the capacity of transmission lines. Instead of assuming that both lines between n_1 and n_2 have infinite capacity, we now assume they just have equal capacity. Then, the same transformation π from the theorem represents the formulation symmetry, and all transformations done by the respective σ from the proof still have to be applied. The only addition is that σ has to account for the capacity constraints. This addition is as follows. Given the constraints $f_{(n_1,n_2),t} \leq ECAP_{(n_1,n_2)}$ and $f_{(n_2,n_1),t} \leq ECAP_{(n_2,n_1)}$ for all $t \in T$, σ swaps the rows of these constraints. Since the capacity $ECAP_{(n_1,n_2)}$ is equal to $ECAP_{(n_2,n_1)}$, this does not change the right-hand side of the constraints, and the composition with the permutation π ensures that the variables on the left-hand side of the problem are in their original rows and columns.

Non-Formulation Symmetry

The assumption that the capacities of both transmission lines is equal is required for proving a formulation symmetry, but without this assumption the problem is still symmetric. As long as there is a transmission line between the two nodes through which electricity can flow, the demand of either countries can partly be satisfied by the production of the other node. Thus, as long as the capacity of the lines is not fully used part of the production of both nodes can be exchanged without changing the objective value.

The same idea holds for the demands of both nodes. These do not have to be equal at every time step to obtain symmetry, not even at a single time step. Since the production and investment costs are the same in both nodes, the production can be done in either of the nodes and transmitted to the other without changing the total costs. The only requirements are that the transmission capacity is not exceeded and that both plants do not produce more than their invested capacity.

Another symmetry which cannot be determined from the formulation is obtained by removing one more assumption, which is the assumption that the set of plants is identical in both nodes. For a symmetry to exist this does not necessarily have to hold for the full set of power plants. Even if one power plant in one of the nodes has the same costs as one plant in the other node, there might be symmetry. As long as these identical plants are profitable enough that it is best to invest in them, production of both plants can be exchanged, regardless of what other plants in these nodes produce. Again this is subject to the transmission capacities and demands, but exchanging production in principal does not lead to extra total costs.

The difficulty with all these non-formulation symmetries is that they are hard to detect from the problem input and specification. They might only be known to exist after the solutions have been found. This is also the reason that solvers do not detect and deal with such symmetries. However, since we have knowledge of the problem which a solver does not have, we know that these symmetries can exist and can deal with them regardless.

7.2. Resolving Symmetries in the Generation Expansion Planning Problem

Given the two symmetries identified in the previous section, we now develop methods for reducing the generation expansion planning problem size using each of the symmetries, as well as algorithms to retrieve an optimal solution for the original problem from the reduced problem outcome. The complete procedure to obtain an optimal solution to the generation expansion problem then consists of three stages: reducing the problem size, solving the problem using a linear programming solver and running the unfolding algorithm.

In Subsection 7.2.1 we propose a reduction method and retrieval algorithm for the symmetry within a node, which was described in 7.1.1. Afterwards, we provide the same methods for the second type of symmetry in Subsection 7.2.2.

7.2.1. Interchangeable Power Plants within a Node

For the type of symmetry introduced by Theorem 7.1.1, we propose to simply combine the equivalent power plants into one to obtain a problem with fewer plants. After solving the smaller problem, we redistribute the production and investments equally among the original plants of each merged generator. The reduced problem then consists of fewer power plan and redistribute the production and investments after solving the reduced problem. Figure 7.1 visualises this approach for a single node with three generators. In the first step, the two symmetric generators are merged and in the last step, the energy is redistributed. The intermediate step represents the solving process of the reduced linear program.

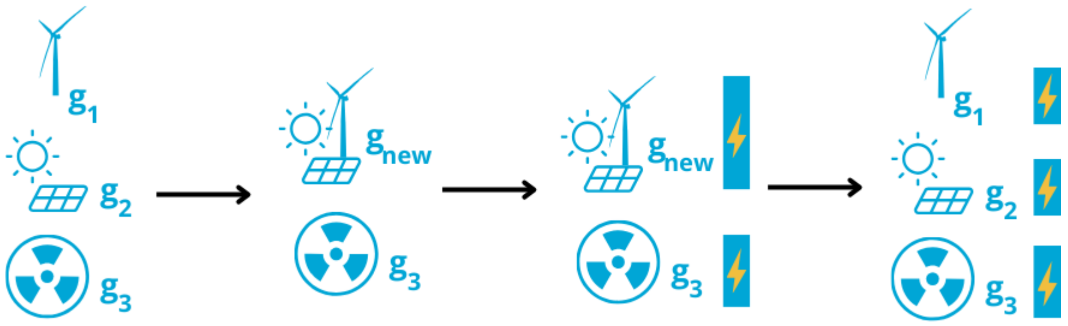


Figure 7.1: Illustration of resolving symmetry between power plants within a node. In the first step, the two symmetric plants g_1 and g_2 are merged to reduce the problem. The second step represents the solver solving this reduced problem. In the third step, we redistribute the energy of the merged plant g_{new} over the original plants g_1 and g_2.

To explain our method in detail, assume there are two power plants $g_1 = (n, e_1)$ and $g_2 = (n, e_2)$ in the same node n and suppose all conditions of Theorem 7.1.1 hold. Before solving the model we apply the following reduction. We create a new plant $g_{new} = (n, e_{new})$ which functions as an aggregation of g_1 and g_2. As a consequence of this, all decision variables involving g_1 and g_2 are replaced by one decision variable for g_{new}, so we obtain $ui_{g_{new}}$ and $p_{g_{new},t}$ for all $t \in \mathcal{T}$. Furthermore, for g_{new}, we have $UCAP_{g_{new}} = UCAP_{g_1}$, $IC_{g_{new}} = IC_{g_1}$, $PC_{g_{new}} = PC_{g_1}$ and $GA_{g_{new},t} = GA_{g_{1},t}$ for all $t \in \mathcal{T}$. The reason for the capacity, costs and availability of g_{new} to be equal to those of g_1 and g_2 instead of the sum of both, is that producing in g_{new} represents producing in either g_1 or g_2, not in both. For the ramping constraints, on the other hand, we cannot copy the constraints of one of the plants, since the difference between time steps of a merged generator can be twice the difference of a single power plant.

If there are more than two symmetric power plants in a node, we simply replace all plants by g_{new} and this process is repeated for all nodes of the original problem. We denote the set of all non-symmetric and merged power plants by $\mathcal{G}'$. Ramping of a merged generator should then be bounded by the sum of the bounds of all original plants. Due to the symmetry in the original generators, the sum of the bounds is equal to the bound of one plant multiplied by the number of plants merged into g_{new}, which we denote by $SUB_{g_{new}}$. The resulting reduced problem is given by (7.5).

$$
\begin{aligned}
\text{minimize} \quad & \sum_{g \in \mathcal{G}'} IC_g \cdot UCAP_g \cdot ui_g + WOP \cdot \left(\sum_{g \in \mathcal{G}', t \in \mathcal{T}} (PC_g \cdot p_{g,t}) + \sum_{n \in \mathcal{N}, t \in \mathcal{T}} (MDC \cdot md_{n,t}) \right) && && (7.5a) \\
\text{subject to} \quad & && && \\
p_{g,t} &\leq GA_{g,t} \cdot UCAP_g \cdot ui_g && \forall g \in \mathcal{G}', t \in \mathcal{T}, && (7.5b) \\
D_{n,t} &= \sum_{g \in \{g' \in \mathcal{G}' | PN_{g'} = n} p_{g,t} + \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = n} f_{l,t} - \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = n} f_{l,t} && && \\
&\quad + md_{n,t} && \forall n \in \mathcal{N}, t \in \mathcal{T}, && (7.5c) \\
f_{l,t} &\geq -ICAP && \forall l \in \mathcal{L}, t \in \mathcal{T}, && (7.5d) \\
f_{l,t} &\leq ECAP && \forall l \in \mathcal{L}, t \in \mathcal{T}, && (7.5e) \\
p_{g,t} - p_{g,t-1} &\geq -R \cdot UCAP_g \cdot ui_g \cdot |SUB_g| && \forall g \in \mathcal{G}', t \in \mathcal{T} \setminus \{1\}, && (7.5f) \\
p_{g,t} - p_{g,t-1} &\leq \ \ R \cdot UCAP_g \cdot ui_g \cdot |SUB_g| && \forall g \in \mathcal{G}', t \in \mathcal{T} \setminus \{1\}, && (7.5g) \\
p_{g,t} &\geq 0 && \forall g \in \mathcal{G}', t \in \mathcal{T}, && (7.5h) \\
md_{n,t} &\geq 0 && \forall n \in \mathcal{N}, t \in \mathcal{T}, && (7.5i) \\
md_{n,t} &\leq D_{n,t} && \forall n \in \mathcal{N}, t \in \mathcal{T}, && (7.5j) \\
ui_g &\geq 0 && \forall g \in \mathcal{G}'. && (7.5k)
\end{aligned}
$$

After solving (7.5), the result has to be converted back to a solution to (3.1) (without the integrality constraint of (3.11)). If one disregards the ramping constraints, this can be done in different ways. One can either choose to assign the full investment $ui_{g_{new}}$ to one of the original generators, say g_1, or split it equally among all power plants in $SUB_{g_{new}}$. Depending on this decision, the same should be done to all production values $p_{g_{new},t}$ for all $t \in \mathcal{T}$. However, to satisfy the original ramping constraints, we cannot assign all investments and production to a single generator, because then there is a possibility that there are two subsequent time steps for which the difference in production for this generator is $|SUB_{g_{new}}|$ times the maximum allowed value. Therefore, we apply an equal split between all original power plants. Pseudocode for this algorithm is given by Algorithm 1.

Algorithm 1 Unfolding algorithm for obtaining a solution to the relaxed version of (3.1), given a solution to (7.5).

```
for $g \in \mathcal{G}'$ do
    for $g_{sub} \in SUB_g$ do
        $ui_{g_{sub}} \leftarrow \frac{ui_g}{|SUB_g|}$
        for $t \in \mathcal{T}$ do
            $p_{g_{sub},t} \leftarrow \frac{p_{g,t}}{|SUB_g|}$
        end for
    end for
end for
```

Note that the decision variables of (7.5) that are unaffected by Algorithm 1 already obtain an optimal value when solving (7.5). We now show that the method proposed in this section returns an optimal solution to the original problem by proving Theorem 7.2.1.

Theorem 7.2.1. Applying Algorithm 1 to an optimal solution of (7.5) returns an optimal solution to the relaxed problem corresponding to (3.1).

Proof. To prove that the proposed method retrieves an optimal solution to the relaxed version of (3.1), we need to show two things. First, we show that given an optimal solution to (3.1), there exists a feasible solution to (7.5) with the same optimal value. Second, we show that given an optimal solution to (7.5), Algorithm 1 finds a feasible solution to (3.1) with the same objective value. If both conditions hold, this means that the optimal objective values of both problems are the same and, therefore, that the proposed method finds the optimal value to the original problem.

For the first condition, let us assume that we have an optimal solution to an arbitrary instance of (3.1) without the integrality constraint of (3.1l), where node n_1 has exactly two plants g_1 and g_2 that are symmetric as described in Theorem 7.1.1. This solution has the objective value of

$$\begin{aligned} C^* &= \sum_{g\in\mathcal{G}} IC_g \cdot UCAP_g \cdot ui_g + WOP \cdot \left(\sum_{g\in\mathcal{G},t\in\mathcal{T}} (PC_g \cdot p_{g,t}) + \sum_{n\in\mathcal{N},t\in\mathcal{T}} (MDC \cdot md_{n,t}) \right) \\ &= IC_{g_1} \cdot UCAP_{g_1} \cdot ui_{g_1} + IC_{g_2} \cdot UCAP_{g_2} \cdot ui_{g_2} + \sum_{g\in\mathcal{G}\setminus\{g_1,g_2\}} IC_g \cdot UCAP_g \cdot ui_g \\ &+ WOP \cdot \left(\sum_{t\in\mathcal{T}} (PC_{g_1} \cdot p_{g_1,t} + PC_{g_2} \cdot p_{g_2,t}) + \sum_{g\in\mathcal{G}\setminus\{g_1,g_2\},t\in\mathcal{T}} (PC_g \cdot p_{g,t}) + \sum_{n\in\mathcal{N},t\in\mathcal{T}} (MDC \cdot md_{n,t}) \right). \end{aligned} \tag{7.6}$$

Now, consider the reduced problem, in which n_1 contains only one power plant g_{new} with the same prices and capacities as both g_1 and g_2. We show that the solution with all unaffected decision variables staying the same, $ui_{g_{new}} = ui_{g_1} + ui_{g_1}$ and $p_{g_{new},t} = p_{g_1,t} + p_{g_2,t}$ for all $t \in \mathcal{T}$, is feasible to the reduced problem and has the same objective value.

To show feasibility, we only have to show that the constraints involving g_{new} are satisfied, as the others remain unchanged. Constraint 3.1b holds, since

$$\begin{aligned} p_{g_{new},t} = p_{g_1,t} + p_{g_2,t} &\leq GA_{g_1,t} \cdot UCAP_{g_1} \cdot ui_{g_1} + GA_{g_2,t} \cdot UCAP_{g_2} \cdot ui_{g_2} \\ &= GA_{g_{new},t} \cdot UCAP_{g_{new}} \cdot ui_{g_{new}} \quad \forall t \in \mathcal{T}. \end{aligned} \tag{7.7}$$

Furthermore, it is clear that Constraint 3.1c holds, since $p_{g_{new},t} = p_{g_1,t} + p_{g_2,t}$ for all $t \in \mathcal{T}$ and g_{new} is in the same node n_1 as g_1 and g_2. Next, to show that the ramping constraints of (3.1f) and (3.1g) are still met, we can apply the same principle as in (7.7). Finally, it is trivial that the bounds of (3.1h) and (3.1k) are still met, because each of these decision variables for g_{new} is the sum of two nonnegative numbers.

Given that we obtained a feasible solution, we show that its objective value is equal to C^*. This derivation is as follows, starting with C' as the objective value of the feasible solution.

$$\begin{aligned} C' &= \sum_{g\in\mathcal{G}'} IC_g \cdot UCAP_g \cdot ui_g + WOP \cdot \left(\sum_{g\in\mathcal{G}',t\in\mathcal{T}} (PC_g \cdot p_{g,t}) + \sum_{n\in\mathcal{N},t\in\mathcal{T}} (MDC \cdot md_{n,t}) \right) \\ &= IC_{g_{new}} \cdot UCAP_{g_{new}} \cdot ui_{g_{new}} + \sum_{g\in\mathcal{G}'\setminus\{g_{new}\}} IC_g \cdot UCAP_g \cdot ui_g \\ &+ WOP \cdot \left(\sum_{t\in\mathcal{T}} (PC_{g_{new}} \cdot p_{g_{new},t}) + \sum_{g\in\mathcal{G}'\setminus\{g_{new}\},t\in\mathcal{T}} (PC_g \cdot p_{g,t}) + \sum_{n\in\mathcal{N},t\in\mathcal{T}} (MDC \cdot md_{n,t}) \right) \end{aligned} \tag{7.8}$$

In the above, $\mathcal{G}'$ is the set of technologies for the reduced problem, containing g_{new} instead of g_1 and g_2.

Since $p_{g_{new},t}$ and $ui_{g_{new}}$ are both sums, we can rewrite C' as follows.

$$\begin{aligned}
C' &= IC_{g_{new}} \cdot UCAP_{g_{new}} \cdot ui_{g_1} + IC_{g_{new}} \cdot UCAP_{g_{new}} \cdot ui_{g_2} + \sum_{g \in \mathcal{G} \setminus \{g_1, g_2\}} IC_g \cdot UCAP_g \cdot ui_g \\
&+ WOP \cdot \left(\sum_{t \in \mathcal{T}} (PC_{g_{new}} \cdot p_{g_1,t} + PC_{g_{new}} \cdot p_{g_2,t}) \right. \\
&+ \left. \sum_{g \in \mathcal{G} \setminus \{g_1, g_2\}, t \in \mathcal{T}} (PC_g \cdot p_{g,t}) + \sum_{n \in \mathcal{N}, t \in \mathcal{T}} (MDC \cdot md_{n,t}) \right) \\
&= IC_{g_1} \cdot UCAP_{g_1} \cdot ui_{g_1} + IC_{g_2} \cdot UCAP_{g_2} \cdot ui_{g_2} + \sum_{g \in \mathcal{G} \setminus \{g_1, g_2\}} IC_g \cdot UCAP_g \cdot ui_g \\
&+ WOP \cdot \left(\sum_{t \in \mathcal{T}} (PC_{g_1} \cdot p_{g_1,t} + PC_{g_2} \cdot p_{g_2,t}) \right. \\
&+ \left. \sum_{g \in \mathcal{G} \setminus \{g_1, g_2\}, t \in \mathcal{T}} (PC_g \cdot p_{g,t}) + \sum_{n \in \mathcal{N}, t \in \mathcal{T}} (MDC \cdot md_{n,t}) \right) = C^*.
\end{aligned} \tag{7.9}$$

Thus, we have shown the first condition of our proof. To show the second condition, suppose we have a solution to the modified problem. We can write the objective value of this solution as follows.

$$\begin{aligned}
C^*_{new} &= \sum_{g \in \mathcal{G}'} IC_g \cdot UCAP_g \cdot ui_g + WOP \cdot \left(\sum_{g \in \mathcal{G}', t \in \mathcal{T}} (PC_g \cdot p_{g,t}) + \sum_{n \in \mathcal{N}, t \in \mathcal{T}} (MDC \cdot md_{n,t}) \right) \\
&= IC_{g_{new}} \cdot UCAP_{g_{new}} \cdot ui_{g_{new}} + \sum_{g \in \mathcal{G}' \setminus \{g_{new}\}} IC_g \cdot UCAP_g \cdot ui_g \\
&+ WOP \cdot \left(\sum_{t \in \mathcal{T}} (PC_{g_{new}} \cdot p_{g_{new},t}) + \sum_{g \in \mathcal{G}' \setminus \{g_{new}\}, t \in \mathcal{T}} (PC_g \cdot p_{g,t}) + \sum_{n \in \mathcal{N}, t \in \mathcal{T}} (MDC \cdot md_{n,t}) \right)
\end{aligned} \tag{7.10}$$

Now, if we transform g_{new} back into g_1 and g_2 according to the algorithm described in this section, we obtain the following objective.

$$\begin{aligned}
C &= \sum_{g \in \mathcal{G}} IC_g \cdot UCAP_g \cdot ui_g + WOP \cdot \left(\sum_{g \in \mathcal{G}, t \in \mathcal{T}} (PC_g \cdot p_{g,t}) + \sum_{n \in \mathcal{N}, t \in \mathcal{T}} (MDC \cdot md_{n,t}) \right) \\
&= IC_{g_1} \cdot UCAP_{g_1} \cdot ui_{g_1} + IC_{g_2} \cdot UCAP_{g_2} \cdot ui_{g_2} + \sum_{g \in \mathcal{G} \setminus \{g_1, g_2\}} IC_g \cdot UCAP_g \cdot ui_g \\
&+ WOP \cdot \left(\sum_{t \in \mathcal{T}} (PC_{g_1} \cdot p_{g_1,t} + PC_{g_2} \cdot p_{g_2,t}) + \sum_{g \in \mathcal{G} \setminus \{g_1, g_2\}, t \in \mathcal{T}} (PC_g \cdot p_{g,t}) + \sum_{n \in \mathcal{N}, t \in \mathcal{T}} (MDC \cdot md_{n,t}) \right)
\end{aligned} \tag{7.11}$$

Since our algorithm preserves the costs and capacities in the transformation, we can rewrite (7.11) as

$$\begin{aligned}
C &= IC_{g_{new}} \cdot UCAP_{g_{new}} \cdot ui_{g_1} + IC_{g_{new}} \cdot UCAP_{g_{new}} \cdot ui_{g_2} + \sum_{g \in \mathcal{G} \setminus \{g_1, g_2\}} IC_g \cdot UCAP_g \cdot ui_g \\
&+ WOP \cdot \left(\sum_{t \in \mathcal{T}} (PC_{g_{new}} \cdot p_{g_1,t} + PC_{g_{new}} \cdot p_{g_2,t}) \right. \\
&+ \left. \sum_{g \in \mathcal{G} \setminus \{g_1, g_2\}, t \in \mathcal{T}} (PC_g \cdot p_{g,t}) + \sum_{n \in \mathcal{N}, t \in \mathcal{T}} (MDC \cdot md_{n,t}) \right)
\end{aligned} \tag{7.12}$$

If we now apply the fact that we equally distribute the investment $ui_{g_{new}}$ and the production $p_{g_{new}}$ over g_1 and g_2, we obtain the following objective value.

$$\begin{aligned}
C = {} & IC_{g_{new}} \cdot UCAP_{g_{new}} \cdot \frac{1}{2} ui_{g_{new}} + IC_{g_{new}} \cdot UCAP_{g_{new}} \cdot \frac{1}{2} ui_{g_{new}} + \sum_{g \in \mathcal{G} \setminus \{g_1, g_2\}} IC_g \cdot UCAP_g \cdot ui_g \\
& + WOP \cdot \left(\sum_{t \in \mathcal{T}} (PC_{g_{new}} \cdot \frac{1}{2} p_{g_{new},t} + PC_{g_{new}} \cdot \frac{1}{2} p_{g_{new},t}) + \sum_{g \in \mathcal{G} \setminus \{g_1, g_2\}, t \in \mathcal{T}} (PC_g \cdot p_{g,t}) \right. \\
& \left. + \sum_{n \in \mathcal{N}, t \in \mathcal{T}} (MDC \cdot md_{n,t}) \right) = C^*_{new}
\end{aligned} \tag{7.13}$$

Showing feasibility of our solution can be done similar to what we did for the first condition of this proof. Therefore, the proposed algorithm guarantees to find an optimal solution to the original problem by applying a symmetry reduction. □

7.2.2. Interchangeable Power Plants in Connected Nodes

In this section we propose a symmetry reduction and retrieval method for the type of symmetry described in Subsection 7.1.2. The approach we take for these methods is similar to that of the previous section, but it involves more variables and, therefore, requires more constraints on the reduced problem.

As explained in Subsection 7.1.2, many assumptions on the input data which are required for the formulation symmetry of Theorem 7.1.2 can be released without making the problem fully asymmetric. The aim of our method is to be as general as possible, in the way that it works in as many symmetric scenarios described in Subsection 7.1.2 as possible. As in Subsection 7.1.2, we start with a simple scenario without transmission capacities and extend this step by step to obtain a general symmetry-handling method.

Infinite Transmission Capacity

For the symmetric case with infinite transmission capacity recall the formulation symmetry described by Theorem 7.1.2. Our idea of exploiting this symmetry is to reduce the problem size is to merge two nodes with symmetric power plants into one. Since it does not matter in which of the two nodes energy is produced, we can solve the problem with fewer nodes, reducing the total number of decision variables significantly. After solving the reduced problem, the production and investments can be split among the original nodes equally.

Figure 7.2 shows an illustration of this process. The first step of this figure shows how we reduce two symmetric nodes into one. The second step represents the solving process by the solver. In the last step, the redistribution of energy is done, where each node gets half of the investments and production. We also illustrated what happens with the flow in a single time step. In the last stage, n_2 has a net incoming flow while n_1 has an outgoing flow, so we need to add the flow from n_2 to n_1 to account for this.

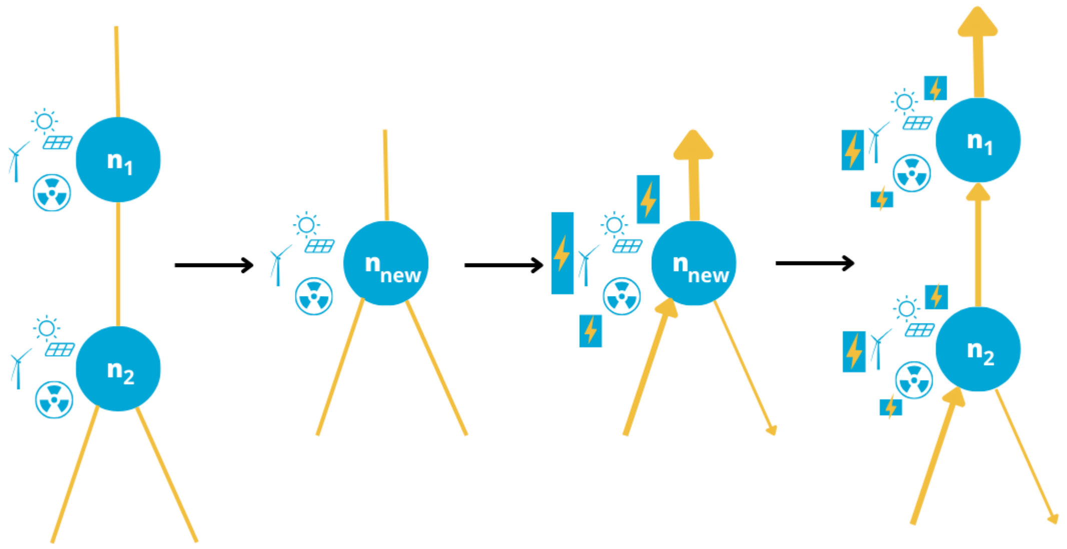


Figure 7.2: Illustration of resolving symmetry between power plants in different nodes. In the first step, the two symmetric nodes n_1 and n_2 are merged to reduce the problem. The second step represents the solver solving this reduced problem. In the third step, we equally redistribute the energy of the merged node n_{new} over the original nodes g_1 and g_2. In this step we also add a flow from n_2 to n_1 to account for the higher net incoming flow in n_2.

In more detail, the method works as follows. Suppose that two nodes n_1 and n_2 in a problem instance have identical sets of power plants, equal demand and a transmission line between them with infinite capacity. These two nodes satisfy all conditions of Theorem 7.1.2, so there is a formulation symmetry. To reduce the problem, we replace n_1 and n_2 by a new node n_{new}. This new node is assigned the sum of the demands of n_1 and n_2, to make sure the total demand in the system remains the same. The power plants located at n_{new}, which is the set $\{g_{new,1}, \ldots, g_{new,m}\}$, have the same unit capacity, investment cost, production cost and generation availability profile as the power plants of n_1 (which is the same as n_2). This is for the same reasons as in the previous subsection, which is that the production of plants in n_{new} represents producing in either n_1 or n_2. Finally, we should also take into account the transmission lines of n_1 and n_2. Clearly, if we merge n_1 and n_2 the transmission line between these two nodes is removed. For the other transmission lines connected to either of these nodes, we connect them to n_{new}. By doing this, we keep the information that n_1 or n_2 has to transmit energy to other nodes or receives energy from other nodes. If n_1 and n_2 connect to the same node n_i, we of course replace both transmission lines by one line between n_{new} and n_i.

After the reduced model described above has been solved, we transform that solution to one for the original problem. Similar to in Section 7.2.1, this can be done in multiple ways. Since the transmission line between n_1 and n_2 has infinite capacity and our model does not factor in transmission costs or loss, we can either do all investment and production in n_1 or split equally or randomly between the two nodes. We again decide to split the investment and production values equally among n_1 and n_2. This is however not the only thing we should do. We should also account for the missed demand $md_{n_{new}}$ and the changes we made in the transmission lines. The missed demand is just equally split among n_1 and n_2. Handling the lines works as follows. The lines that connect n_{new} to other nodes are transformed back in the lines that they originally represented, taking the determined flow. So if, for example, a line which was originally from n_i to n_1, now obtains a flow of f as a line from n_i to n_{new}, we turn this back into a line from n_i to n_1 with flow f. This is done for each time step. Only if we have a line that was originally two lines to the same node, we distribute the flows equally. So then the flows of n_1 and n_2 to this node will each be half of the flow from n_{new} to that node. Lastly, we have to recreate the line that was originally between n_1 and n_2. For this line we have to compute the flow based on the production and incoming and outgoing flows of n_1 and n_2. Since we decided to split the production equally between n_1 and n_2 and their demands are equal, the flow in the line between n_1 and n_2 is the difference between the net incoming flow of both nodes. If n_1 has a higher net incoming flow than n_2, for example, we need to transmit energy from n_1 to n_2 to satisfy the demand of n_2.

Limited Transmission Capacity

The previous symmetry reduction method does not apply to real-world scenarios. In case transmission capacities are enforced, the reduced problem might find a better solution than actually possible. To illustrate this, we provide an example in Figure 7.3, which displays the flow and production decisions for a single time step for both the original and reduced problem.

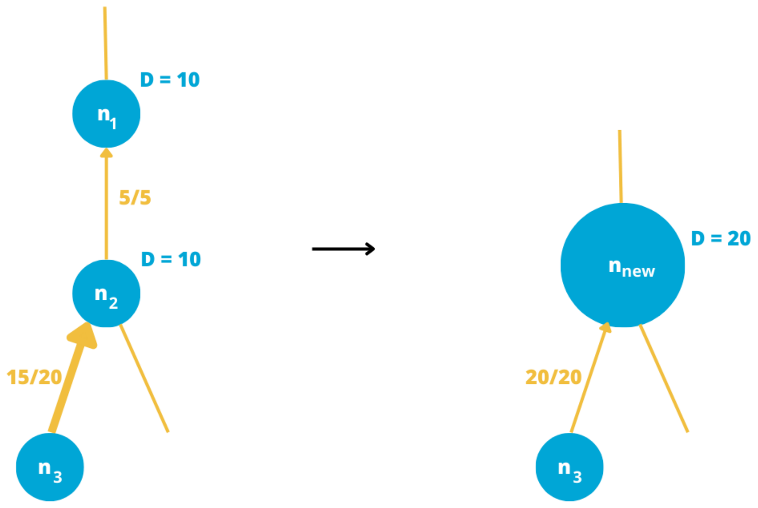


Figure 7.3: Counterexample for why the symmetry resolving method of Subsection 7.2.2 does not work when transmission lines have capacities. The left-hand side portrays an instance of the original problem where n_1 and n_2 are symmetric and n_3 has cheaper power plants. The demand of n_1 and n_2 is 10MW and the flows and capacities are given next to the lines. On the right-hand side n_{new} is the merged node of n_1 and n_2 corresponding to the reduction of Subsection 7.2.2. Here the total flow entering n_{new} is higher than what is possible for the original problem.

On the left-hand side of Figure 7.3, there are three nodes of which n_1 and n_2 are symmetric, i.e. they have symmetric power plants and the same demand profile. The third node n_3 is considered to be cheaper than both n_1 and n_2. Furthermore, the example assumes a demand of 10 MW for both n_1 and n_2 and a transmission capacity of at most 5 MW from n_2 to n_1 and at most 20 MW from n_3 to n_2. Given that n_3 is cheaper, the optimal outcome would be to produce as much of the demand of n_1 and n_2 as possible by power plants in n_3. So, for the original problem this means transporting 15 MW from n_3 to n_2 of which 5 MW is transported further to n_1. It is impossible to produce all of the demand of n_1 using plants in n_3 since the transmission line between n_2 and n_1 is a bottleneck.

The right-hand side of Figure 7.3 shows the reduced problem if we merge the nodes. On that side, n_1 and n_2 have been replaced by n_{new} with a demand equal to the sum of the demands of n_1 and n_2, being 20MW. Now, if we optimise this problem, we will obtain an outcome in which the total demand of n_{new} is produced by n_3 and transported to n_{new}. This results in a better objective value than in the original problem, because we assumed producing in n_3 is cheaper. Therefore, the proposed reduction algorithm does not preserve optimality whenever transmission capacities are taken into account.

The reason that the algorithm does not work anymore, is the fact that we lose information by removing the edge between n_1 and n_2. To resolve this issue, we have to keep some of the variables we intended to remove. Instead of merging two nodes and their power plants completely, we therefore only merge the production values of one pair of symmetric power plants per two nodes. Furthermore, we need to add a set of extra constraints.

In detail, the merging process now works as follows. Suppose the power plants g_1 and g_2 of nodes n_1 and n_2 are symmetric, so g_1 and g_2 share the same costs, capacities and availability profile. If this is the case, we define a new node n_{new} and create a new power plant g_{new} located at n_{new} with the same costs, capacities and availability profile as g_1 and g_2. The other plants of n_1 and n_2 will also be considered to be plants of n_{new} in the reduced problem. Note that we do not require the demand profiles of n_1 and n_2 to be the same, nor do we require the transmission capacity from n_1 to n_2 to be equal to the capacity in the opposite direction. Therefore, the method we develop applies to non-formulation symmetries, instead of the formulation symmetry described by Theorem 7.1.1.

To account for these changes in the merging procedure, we need to adjust (3.1b) and (3.1c). Instead of (3.1b), we now use

$$p_{g,t} \leq \sum_{g_{sub} \in SUB_g} GA_{g_{sub},t} \cdot UCAP_{g_{sub}} \cdot ui_{g_{sub}} \quad \forall g \in \mathcal{G}', t \in \mathcal{T}, \tag{7.14}$$

where $\mathcal{G}'$ is the set of all power plants in the reduced problem, and SUB_g consists of the plants for which the production is combined into the production of g. If g remains unchanged in the symmetry reduction, SUB_g solely consists of g. Thus, the modification of the constraint ensures that the production of a merged plant is bounded by the sum of the production capacities of the plants it originates from.

We replace (3.1c) by

$$\begin{aligned} &\sum_{g \in \{g' \in \mathcal{G}' | PN_{g'} = n\}} p_{g,t} + \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} \in SUBN_n\}} f_{l,t} - \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} \in SUBN_n\}} f_{l,t} \\ &+ \sum_{n_{sub} \in SUBN_n} md_{n_{sub},t} = \sum_{n_{sub} \in SUBN_n} D_{n_{sub},t} \quad \forall n \in \mathcal{N}', t \in \mathcal{T}, \end{aligned} \tag{7.15}$$

where $\mathcal{N}'$ consists of all nodes in the reduced problem and $SUBN_n$ is the set of nodes merged into n. If $n \in \mathcal{N}'$ is a node of the original problem, the constraint is exactly equal to (3.1c). Otherwise, n originates from two nodes and the sum of the production done in both nodes, the flows coming into and out of the nodes and the missed demand of both nodes is equal to the sum of their demands. Note that the flow between the two nodes is both added and subtracted, so not taken into account.

After all these changes, there are still two possible causes for the reduced problem to find solutions that are infeasible to the original problem. The first cause is that the non-merged plants of a node n_1 produce more than the total energy used in n_1 or transported out of n_1. This can occur since (7.15) only ensures that the total energy used in both n_1 and n_2 is equal to the total demand of both nodes. To prevent this, we introduce the following two sets of constraints.

$$\begin{aligned} &\sum_{g \in \{g' \in \mathcal{G}' | PN_{SUB_{g'}} = \{SUBN_{n,1}\}\}} p_{g,t} + \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = SUBN_{n,1}\}} f_{l,t} \\ &- \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = SUBN_{n,1}\}} f_{l,t} + md_{SUBN_{n,1},t} \qquad \leq D_{SUBN_{n,1}} \forall n \in \mathcal{N}_{new}, \forall t \in \mathcal{T}, \end{aligned} \tag{7.16}$$

$$\begin{aligned} &\sum_{g \in \{g' \in \mathcal{G}' | PN_{SUB_{g'}} = \{SUBN_{n,2}\}\}} p_{g,t} + \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = SUBN_{n,2}\}} f_{l,t} \\ &- \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = SUBN_{n,2}\}} f_{l,t} + md_{SUBN_{n,2},t} \qquad \leq D_{SUBN_{n,2}} \forall n \in \mathcal{N}_{new}, \forall t \in \mathcal{T}. \end{aligned} \tag{7.17}$$

In these constraints, $\mathcal{N}_{new}$ contains all nodes that are created in the reduction and PN_{SUB_g} refers to the set of original nodes of a plant g in the reduced problem. This set thus contains either only PN_g, if g is not a merged node, or both $PN_{SUB_{g,1}}$ and $PN_{SUB_{g,2}}$ if g is merged from SUB_{g_1} and SUB_{g_2}.

The second possible problem is that the production value of a merged plant is too big for the capacity of one of the two original plants. This could be the case, for instance, whenever the inequality of (7.16) becomes an equality for n_1. In that scenario, all production done in g_{new} should represent production done by g_2. However, (7.14) only states that $p_{g_{new},t}$ is bounded by the sum of the production capacities of g_1 and g_2. Therefore, it might occur that $p_{g_{new},t}$ is bigger than the production capacity of g_2, creating an infeasible solution to the original problem.

To resolve this, we need to formulate constraints similar to (7.16) and (7.17), but with the production of the merged plant added on the left-hand side and the production capacity of the other node added to the right-hand side. Therefore, we obtain

$$
\begin{aligned}
&p_{g,t} + \sum_{g_1 \in \{g' \in \mathcal{G}' | PN_{SUB_{g'}} = \{PN_{SUB_{g,1}}\}\}} p_{g_1,t} + \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = SUBN_{n,1}\}} f_{l,t} \\
&- \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = SUBN_{n,1}\}} f_{l,t} + md_{SUBN_{n,1},t} \\
&\leq D_{SUBN_{n,1}} + GA_{SUB_{g,2},t} \cdot UCAP_{SUB_{g,2}} \cdot ui_{SUB_{g,2}} && \forall g \in \mathcal{G}_{new}, \forall t \in \mathcal{T}, \quad (7.18) \\
&p_{g,t} + \sum_{\{g' \in \mathcal{G}' | PN_{SUB_{g'}} = \{PN_{SUB_{g,2}}\}\}} p_{g',t} + \sum_{\{l \in \mathcal{L} | NB_l = SUBN_{n,2}\}} f_{l,t} \\
&- \sum_{\{l \in \mathcal{L} | NA_l = SUBN_{n,2}\}} f_{l,t} + md_{SUBN_{n,2},t} \\
&\leq D_{SUBN_{n,2}} + GA_{SUB_{g,1},t} \cdot UCAP_{SUB_{g,1}} \cdot ui_{SUB_{g,1}} && \forall g \in \mathcal{G}_{new}, \forall t \in \mathcal{T}. \quad (7.19)
\end{aligned}
$$

The problem we resolved using these constraints also motivates why we do not merge multiple pairs of symmetric plants in a pair of nodes. If we namely reduce two pairs of plants in a single pair of nodes, (7.18) and (7.19) do not suffice. In that situation, we introduce the problem that the sum of the production values of both reduced plants can become bigger than the sum of the energy used in n_1 and the production capacity of n_2, which is impossible in the original problem.

This problem would require us to add constraints like (7.18) and (7.19), replacing $p_{g,t}$ by the sum of the production of two power plants and the production capacity on the right-hand side by the sum of the production capacities of two sub-plants. Although this would resolve all problems, we would require even more constraints if we merge more than two plants. In fact, for every subset of the merged plants in a node, we require an extra constraint. The runtime likely explodes due to the exponential increase in number of constraints based on the number of merged plants, so we decide to only merge one pair of plants per pair of nodes.

The full formulation of the linear program we end up with is given in Appendix A.1.

Unfolding the Reduced Problem

Since we significantly modified our reduction algorithm, the method for retrieving a solution to the original problem should also be updated. Of course, all variables that have not been changed in the reduction remain unchanged. This leaves us with the production values of reduced power plants that still require a solution value.

To calculate the production of individual power plants we propose a greedy algorithm. For each plant g_{new}, merged from g_1 of n_1 and g_2 of n_2, we produce as much as possible in g_1, and assign the remaining production to g_2. This maximum amount for $p_{g_1,t}$ is the total energy used in n_1 which is its demand subtracted by its excess flow, its missed demand and the production of the non-merged plants in n_1. Due to the constraints of the reduced problem, this value will always be smaller than $p_{g_{new},t}$ and the production capacity of g_1.

The production of g_1 in node n_1 is, therefore, equal to

$$
p_{g_1,t} = D_{n_1,t} - md_{n_1,t} + \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = n_1\}} f_{l,t} - \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = n_1\}} f_{l,t} - \sum_{g \in \{g' \in \mathcal{G}' | PN_{SUB_{g'}} = \{PN_{SUB_{g,1}}\}\}} p_{g,t}. \quad (7.20)
$$

We can now use $p_{g_1,t}$ and $p_{g_{new},t}$ to compute $p_{g_2,t}$ for every time step t, since the production of g_{new} is the sum of the productions of g_1 and g_2. This gives us

$$
p_{g_2,t} = p_{g_{new},t} - p_{g_1,t}. \quad (7.21)
$$

This procedure is repeated for each time step $t \in \mathcal{T}$ to obtain a solution to (3.1). Pseudocode for this method is given by Algorithm 2.

Algorithm 2 Unfolding algorithm for obtaining a solution to the relaxed version of (3.1) without ramping, given a solution to (A.1).

```
for $g_{new} \in \mathcal{G}_{new}$ do
    for $t \in \mathcal{T}$ do
        $p_{SUB_{g,1},t} \leftarrow D_{n_1} - md_{n_1} + \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = n_1\}} f_{l,t} - \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = n_1\}} f_{l,t}$
                $- \sum_{g \in \{g' \in \mathcal{G}' | PN_{SUB_{g'}} = \{PN_{SUB_{g,1}}\}\}} p_{g,t}$
        $p_{SUB_{g,2},t} \leftarrow p_{g_{new},t} - p_{SUB_{g,1},t}$
    end for
end for
```

To show that the method proposed in this section is correct, we provide a proof of Theorem 7.2.2 below in Appendix A.2.

Theorem 7.2.2. Applying Algorithm 2 to an optimal solution of (A.1) returns an optimal solution to the relaxed problem corresponding to (3.1) without ramping constraints.

Ramping

It is important to note that when the ramping constraints of (3.1f) and (3.1g) are enforced, the method described in this section might compute a solution infeasible to (3.1), with a better objective value than the optimal solution to (3.1). Dealing with the ramping constraints in the reduced problem is complex, as simply enforcing the constraints

$$p_{g,t} - p_{g,t-1} \geq -R \cdot \sum_{g_{sub} \in SUB_g} UCAP_{g_{sub}} \cdot ui_{g_{sub}} \qquad \forall g \in \mathcal{G}', t \in \mathcal{T} \setminus \{1\}, \tag{7.22}$$

$$p_{g,t} - p_{g,t-1} \leq R \cdot \sum_{g_{sub} \in SUB_g} UCAP_{g_{sub}} \cdot ui_{g_{sub}} \qquad \forall g \in \mathcal{G}', t \in \mathcal{T} \setminus \{1\}, \tag{7.23}$$

is not enough. To show this, we take a look at the example in Figure 7.4. This figure shows a scenario with two nodes n_1 and n_2, in which both nodes have exactly one power plant and these power plants are merged by our method. In this scenario, we consider two time steps. At time step $t = 1$, n_1 has a demand of 10 and n_2 has a demand of 0. At $t = 2$, the demands are swapped. Furthermore, for the clarity of the example, we assume that the maximum production capacity of g_1 in n_1 and g_2 in n_2 is equal to 10. In our model, the maximum capacity is determined by the investment. Finally, we have a ramping parameter of $R = 0.2$.

With all these requirements, the original problem can satisfy at most a demand of 6 out of 10 at both time steps, since both n_1 and n_2 can produce at most 2 when their demand is 0. Therefore, in the other time step, they can produce at most 4. For the reduced problem, we can just satisfy the demand of 10 in both time steps, since the difference between the production in both time steps is 0, so this definitely satisfies both (7.22) and (7.23). Thus, the reduced problem will find a solution with a higher demand satisfaction, so a lower objective value.

To resolve this, we have to add multiple constraints bounding the maximum production at the current time step. To derive these constraints, we consider a merged power plant g_{new} in node n, originating from g_1 in n_1 and g_2 in n_2. We know that the part of the production of g_{new} representing production of g_1 at time step t should be smaller or equal to the sum of the maximum possible production of g_1 at time step $t + 1$ and the maximum ramping for g_1. The maximum possible production at time step $t + 1$ in g_1 is the minimum of the production capacity of g_1 at $t + 1$ and the energy used at n_1 at $t + 1$ minus the energy produced by non-merged nodes. Therefore, we can bound the production of g_{new} at time step t by the sum of this maximum possible production of g_1 at $t + 1$, the maximum ramping for g_1 and the production capacity of g_2 at time step t. We apply the same to $t - 1$ and to g_2 instead of g_1 to obtain the following four sets of constraints.

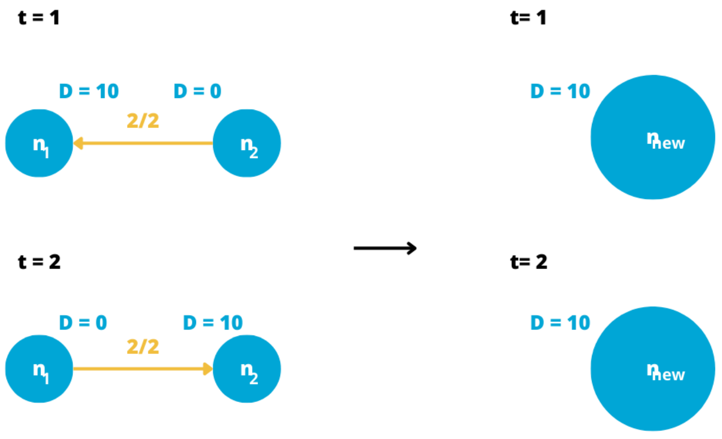


Figure 7.4: Counterexample for why (7.22) and (7.23) are not satisfactory to deal with ramping in the reduced problem. The left-hand side shows two time steps for the original problem, with two symmetric nodes containing a single power plant. In this scenario, it is optimal to produce 4 in n_1 and 2 in n_2 at $t = 1$, and 2 in n_1 and 4 in n_2 at $t = 2$. The right-hand side shows the same situation for the reduced problem, where it is possible to satisfy the full demand at both time steps.

$$
\begin{aligned}
p_{g_{new},t} \leq \min\Bigg(& D_{n_1,t+1} - md_{n_1,t+1} + \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = n_1\}} f_{l,t+1} - \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = n_1\}} f_{l,t+1} \\
& - \sum_{g \in \{g' \in \mathcal{G}' | PN_{SUB_{g'}} = \{g_1\}\}} p_{g,t+1}, GA_{g_1,t+1} \cdot UCAP_{g_1} \cdot ui_{g_1} \Bigg) \\
& + R \cdot UCAP_{g_1} \cdot ui_{g_1} + GA_{g_2,t} \cdot UCAP_{g_2} \cdot ui_{g_2} \qquad \forall t \in \mathcal{T} \setminus \{T\}, \qquad (7.24)
\end{aligned}
$$

$$
\begin{aligned}
p_{g_{new},t} \leq \min\Bigg(& D_{n_1,t+1} - md_{n_1,t-1} + \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = n_1\}} f_{l,t-1} - \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = n_1\}} f_{l,t-1} \\
& - \sum_{g \in \{g' \in \mathcal{G}' | PN_{SUB_{g'}} = \{g_1\}\}} p_{g,t-1}, GA_{g_1,t-1} \cdot UCAP_{g_1} \cdot ui_{g_1} \Bigg) \\
& + R \cdot UCAP_{g_1} \cdot ui_{g_1} + GA_{g_2,t} \cdot UCAP_{g_2} \cdot ui_{g_2} \qquad \forall t \in \mathcal{T} \setminus \{1\}, \qquad (7.25)
\end{aligned}
$$

$$
\begin{aligned}
p_{g_{new},t} \leq \min\Bigg(& D_{n_2,t+1} - md_{n_2,t+1} + \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = n_2\}} f_{l,t+1} - \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = n_2\}} f_{l,t+1} \\
& - \sum_{g \in \{g' \in \mathcal{G}' | PN_{SUB_{g'}} = \{g_2\}\}} p_{g,t+1}, GA_{g_2,t+1} \cdot UCAP_{g_2} \cdot ui_{g_2} \Bigg) \\
& + R \cdot UCAP_{g_2} \cdot ui_{g_2} + GA_{g_1,t} \cdot UCAP_{g_1} \cdot ui_{g_1} \qquad \forall t \in \mathcal{T} \setminus \{T\}, \qquad (7.26)
\end{aligned}
$$

$$
\begin{aligned}
p_{g_{new},t} \leq \min\Bigg(& D_{n_2,t+1} - md_{n_2,t-1} + \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = n_2\}} f_{l,t-1} - \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = n_2\}} f_{l,t-1} \\
& - \sum_{g \in \{g' \in \mathcal{G}' | PN_{SUB_{g'}} = \{g_2\}\}} p_{g,t-1}, GA_{g_2,t-1} \cdot UCAP_{g_2} \cdot ui_{g_2} \Bigg) \\
& + R \cdot UCAP_{g_2} \cdot ui_{g_2} + GA_{g_1,t} \cdot UCAP_{g_1} \cdot ui_{g_1} \qquad \forall t \in \mathcal{T} \setminus \{1\}. \qquad (7.27)
\end{aligned}
$$

In the above, T is the maximum time step in $\mathcal{T}$. Next to that, these constraints should be formulated for each $g_{new} \in \mathcal{G}_{new}$. Also, note that since each of the inequalities contains a minimum of two terms, these constraints should actually be split into two constraints each. Therefore, dealing with ramping constraints requires eight constraints per time step per node, and each of the constraints involves multiple decision variables. Therefore, the advantage of merging power plants might be disregarded by the large increase in number of constraints to handle ramping.

8 Effects of Symmetry Resolving Methods on Crossover

In the previous chapter, we identified two types of symmetry in the generation expansion problem and proposed an algorithm for exploiting each of these. In this chapter, we compare the runtime of both algorithms to the runtime of solely running Gurobi's linear programming solver. To do so, we use the European case study to derive problem instances containing each of the types of symmetry. These instances are described in Section 8.1. The other two sections of this chapter display the results of running both Gurobi's solver and the symmetry resolving methods of Section 7.2 using these instances, where Section 8.2 focuses on the type of symmetry described in Subsection 7.1.1 and 8.3 on the second type of symmetry.

8.1. Symmetric Problem Instances

To examine the effects of the symmetry resolving methods discussed in Section 7.2, we require problem instances containing the type of symmetry the algorithm tries to resolve. We obtain two such instances, one for each type of symmetry, by modifying the input data of the European case study described in Section 3.2.

For the type of symmetry described in Subsection 7.1.1, we develop an instance which contains a formulation symmetry in accordance with Theorem 7.1.1. As this theorem expects power plants within a node to be identical, we set the investment costs of all plants in a node, for instance Austria, to 20000 euros per megawatt, the variable production costs to 0.1 euros per megawatt hour and the capacity of one unit to 500 megawatt. These values make the plants in this node cheaper than the other plants in the input data, so the symmetry occurs in the optimal region of the search space. Furthermore, if a plant uses a renewable energy source, we set its generation availability profile to a flat profile of all ones, to ensure that this plant is symmetric to non-renewable plants.

To determine both the effect of this symmetry on Gurobi's solver and the effectiveness of the symmetry reduction algorithm of Subsection 7.2.1, we create a very symmetric scenario by applying the modification above to all nodes. To prevent creating symmetry between nodes, we slightly change the values for every node. Each node has an investment cost of 1000 euros more, production costs of 0.1 euros extra and its unit capacity is ten megawatts smaller. This is done in alphabetical order, so power plants in Belgium, for instance, have 21000 euros of investment costs, 0.2 euros of variable costs and a unit capacity of 490 MW.

Using this highly-symmetric problem instance, we compare the runtime of each solving phase in Gurobi's solving algorithm and in our symmetry resolving method. Similar to in Chapter 5, we test both methods using a planning horizon of ten to fifty weeks. The expectation is that the longer the planning horizon is, the more effective our symmetry resolving method is, since it reduces the number of production variables by the number of symmetric plants for every time step.

Not only the planning horizon affects the amount of symmetry in a problem instance, the number of identical plants also does. The more plants with identical costs and capacities there are, the more symmetry a problem instance has. We can use this to test whether a more symmetric problem results in a longer crossover time for Gurobi and a larger benefit of our symmetry handling method. To do so, we define the symmetry ratio of a problem instance by the fraction of nodes for which all plants have equal costs and capacities. So, for the European case study this ratio is defined by

$$\text{Symmetry ratio } = \frac{\text{\# symmetric nodes}}{\text{\# total nodes}} = \frac{\text{\# symmetric nodes}}{20}. \tag{8.1}$$

We use this ratio to create five test cases, with symmetry ratios of 0, $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$ and 1, respectively. The first test case contains the original data, the second has five symmetric nodes, etc. The symmetric nodes are added in alphabetical order to ensure that the set of symmetric nodes in each scenario is a subset of the next scenario's set of symmetric nodes. If this were not the case, a scenario with many symmetric nodes could be less symmetric than a scenario with fewer symmetric nodes due to the different numbers of power plants in each node.

Each of these five scenarios are solved using both Gurobi and the symmetry resolving method of Subsection 7.2.1 on a planning horizon of fifty weeks. Section 8.2 shows the results of this and the tests described previously in this section.

For the other type of symmetry, which is described in Subsection 7.1.2, we simply use the symmetric scenario defined in Section 5.2. That scenario contains symmetry as described in Subsection 7.1.2, but not a formulation symmetry. As the aim of the symmetry handling method of Subsection 7.2.1 is able to deal with non-formulation symmetry, this scenario is suitable to test its effects.

It is also interesting to apply our algorithm to the baseline problem instance, since the original input data of the European case study already assumes an equal variable production price if the technology used is the same. Therefore, connected nodes with the same technology can be reduced by our symmetry resolving algorithm. The results of running our algorithm for both the baseline and the symmetric problem are given in Section 8.3.

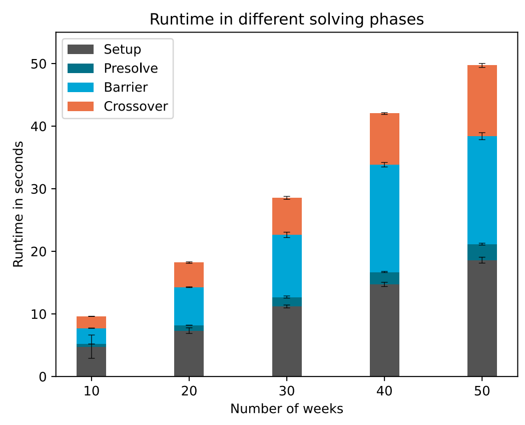
8.2. Interchangeable Power Plants within a Node

As described in the previous section, we run several experiments with a problem instance containing the type of symmetry described in Subsection 7.1.1. The results of each of these are given in this section.

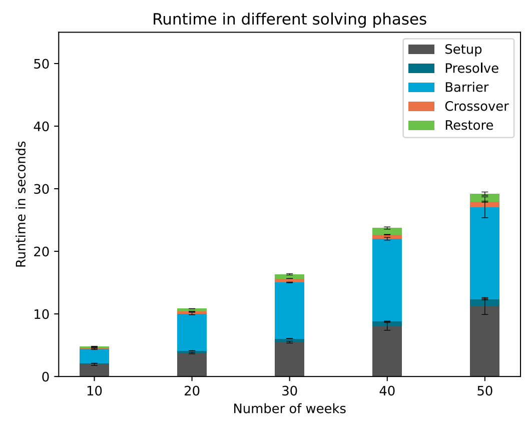
The first set of experiments investigates the effect of symmetry on the runtime of Gurobi's solver and the symmetry reduction algorithm proposed in Subsection 7.2.1. Figure 8.1 shows four plots, each displaying the runtime in seconds of all solving phases for a planning horizon of ten to fifty weeks. Subfigures 8.1a and 8.1b show the results of solving the problem instance without ramping constraints, whereas Subfigures 8.1c and 8.1d display the runtimes with ramping constraints. At the same time, Subfigures 8.1a and 8.1c contain the results of using Gurobi's solver, whereas Subfigures 8.1b and 8.1d use our proposed method.

The results clearly show that our symmetry reduction method is effective in reducing the runtime of crossover. Moreover, the sum of crossover's runtime and the time to unfold the solution is much smaller than the runtime of crossover when our algorithm is not applied. Furthermore, the setup time also decreased due to applying our method. This is likely due to the lower number of constraints and variables that have to be initialised. The results also do not show an increase in the runtime of the barrier or presolving methods, making our algorithm significantly faster than solely using Gurobi to solve such a symmetric problem instance.

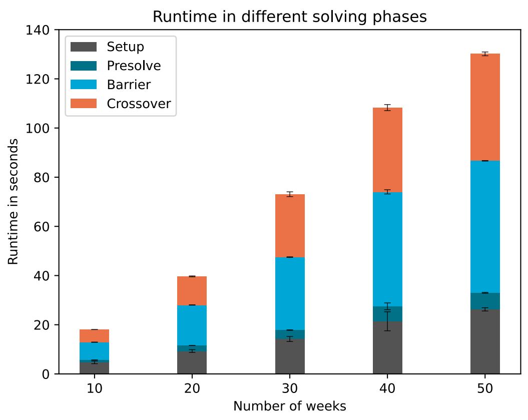
Similar to in Chapter 5, the difference between using ramping constraints or not does not influence the runtime of crossover specifically. The same trend with respect to the planning horizon is visible, only the runtime of each solving phase is longer when ramping constraints are enforced. Also, if we compare the results of Subfigures 8.1a and 8.1c to those in Figure 5.1, we see that the symmetry we introduced



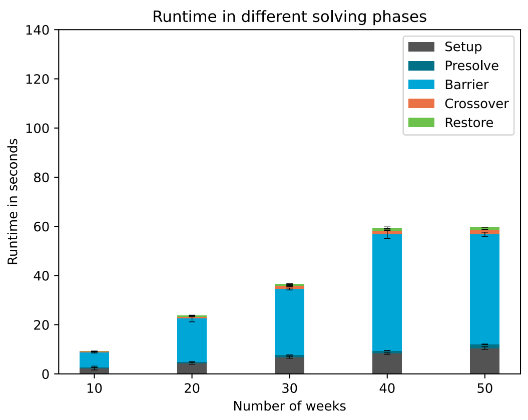
(a) Gurobi without ramping.



(b) Proposed method without ramping.



(c) Gurobi with ramping.



(d) Proposed method with ramping.

Figure 8.1: Runtime of different solving phases for the problem instance containing formulation symmetry according to Theorem 7.1.1, with production horizons of ten to fifty weeks.

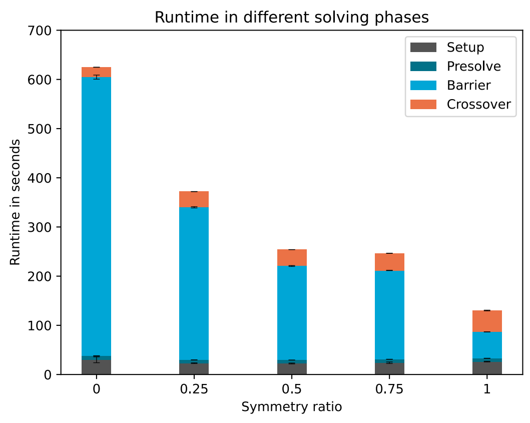
has a significant impact on the runtime of crossover, as the fraction of time spent in crossover is clearly larger for this problem instance.

As the plots of Figure 8.1 only display results for a very symmetric problem instance, it is also interesting to investigate how the amount of symmetry affects both our method and Gurobi's solver. Therefore, Figure 8.2 displays the results of solving problem instances with different symmetry ratios, as defined in (8.1). All runs have been done using a planning horizon of fifty weeks and Subfigures 8.2a and 8.2b display the results of only using Gurobi and using our method, respectively.

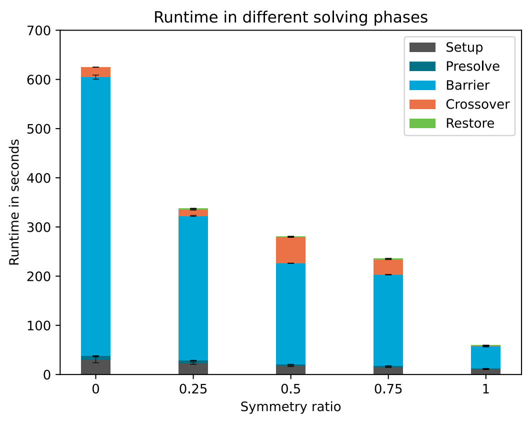
In both subfigures the total runtime decreases when the problem instance contains more symmetry. However, when using Gurobi, the runtime in crossover increases. Therefore, these results clearly show that symmetry benefits the barrier method while it harms the runtime of crossover.

The results in Subfigure 8.2b follow a different pattern. First of all, note that the results for a symmetry ratio of 0 are exactly the same to those of Gurobi, as in that case our problem reduction cannot be applied. For the other symmetry ratios we notice two things. First, we observe that the higher the symmetry ratio is, the smaller the setup time becomes. This is likely due to the fact that fewer variables and constraints are initialised when our algorithm merges more power plants. Second, there is no clear trend in the runtime of crossover as opposed to the results for Gurobi. However, for all ratios except 0.5 (and 0) the average time spent on crossover and unfolding the solution is smaller than Gurobi's runtime in crossover.

Similarly, the total runtime of our method is smaller than that of Gurobi for all symmetry ratios except



(a) Gurobi.



(b) Proposed method.

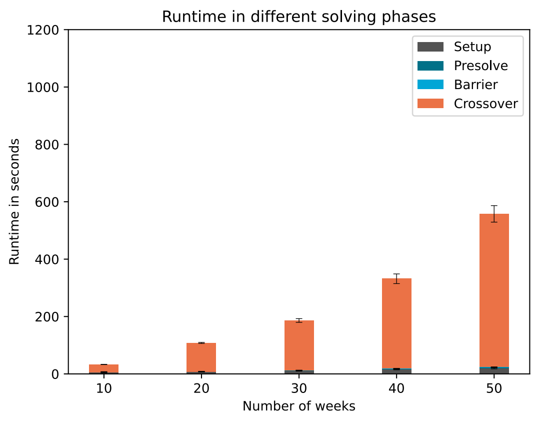
Figure 8.2: Runtime of different solving phases for the problem instance containing formulation symmetry according to Theorem 7.1.1, with ramping constraints and a production horizon of fifty weeks. The x-axis represents the different symmetry ratios, where a higher number corresponds to more symmetry.

0 and 0.5. For symmetry ratios of 0.25 and 1, it is clearly visible that our method is faster. For a ratio of 0.75 we employed a one-tailed two-sample t-test to show that our method (M = 235.95, SD = 0.85) is significantly faster than Gurobi (M = 246.51, SD = 1.62), t(4) = 11.53, p<0.00001. So, our method is often faster than Gurobi whenever symmetry between plants in the same node is present, but more symmetry does not necessarily make our method perform better.

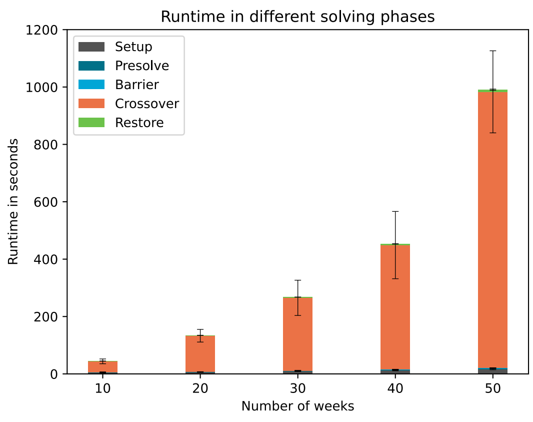
8.3. Interchangeable Power Plants in Connected Nodes

To test the symmetry resolving method of Subsection 7.2.2, we apply this algorithm to the symmetric problem instance described in Section 5.2 and compare the results against those in Figure 5.2. Note that we disregard the ramping constraints for these results, since our unfolding algorithm, Algorithm 2, does not account for such constraints.

Figure 8.3 shows the runtimes of running Gurobi's method and the method proposed in Subsection 7.2.2 on the problem instance containing a lot of symmetry between different nodes. Subfigure 8.3a contains the results of running Gurobi and is a copy of Subfigure 5.2b with a different scale on the y-axis. The other subfigure displays the runtimes when using our proposed method.



(a) Gurobi (copy of 5.2b with different scale on y-axis).



(b) Proposed method.

Figure 8.3: Runtime of different solving phases for the problem instance containing symmetry as described in Subsection 7.2.2, without ramping constraints for a production horizon of fifty weeks.

These results show that, on average, our method is slower than solely using Gurobi. However, the error bars on the runtime of crossover in Subfigure 8.3b have a significantly larger range than those in Subfigure 8.3a. The reason for this is that in each run of our algorithm a different set of power plants can be merged, since we loop through the power plants in a random order when checking whether a pair of plants is symmetric.

For some problem reductions, the runtime of our method was faster than that of Gurobi. To identify how frequently this happens, we solve the same problem instance, with a planning horizon of fifty weeks, thirty times using our proposed algorithm. Table 8.1 summarises the results by displaying how often our proposed algorithm was faster than, slower than or as fast as Gurobi. This is done for each of the solving phases and the total runtime. We consider our algorithm equally fast whenever its runtime is between the slowest and fastest of the five runs we performed previously using Gurobi. Note that we compare Gurobi's crossover runtime to the sum of our method's crossover runtime and its time for recovering the solution to the original problem.

Table 8.1: Results of solving the symmetric problem instance described in Section 5.2 thirty times using our symmetry resolving method. The results indicate how many times our method is faster than, slower than or as fast as solely using Gurobi to solve the instance for each solving phase.

	# times faster	# times slower	# times equally fast
Setup	25 (83.33%)	0 (0%)	5 (16.67%)
Presolve	0 (0%)	30 (100%)	0 (0%)
Barrier	27 (90%)	3 (10%)	0
Crossover + restore	5 (16.67%)	19 (63.33%)	6 (20%)
Total runtime	5 (16.67%)	18 (60%)	7 (23.33%)

Although these results show that our method is faster in only one in six runs and slower in 60% of runs, the faster runs still indicate that our method can be beneficial. Especially the fact that the fastest run of our method is 15.45% faster than the fastest run using Gurobi suggests that finding the right pairs of power plants to merge could benefit the runtime of crossover and solving the whole problem significantly.

Next to this, it is also worthwile noting that the barrier method is faster in 27 out of thirty runs using our method. This could mean that our method reduces symmetry Gurobi cannot detect. The combination of our reduction and Gurobi's symmetry reduction then results in an even smaller problem.

The last experiment we performed was to run the method we proposed on the baseline problem instance of the European case study, to test our method on real-world data. We performed the same experiment as for Figure 5.1b. This resulted in a longer runtime in both barrier and crossover. Moreover, the barrier method terminated early in several cases, leaving crossover with a starting point further away from the optimum. Also, the runtime of both barrier and crossover differed a lot in each of the five runs. Reasons for our method to not perform well on the baseline problem could be a lack of symmetry in the problem and, again, the manner in which power plants are paired up to be merged. More details on these results can be found online at: https://github.com/marnoldus/accelerate-crossover-using-symmetry.

9 Conclusion

The aim of this thesis consisted of tackling two separate challenges. Our first goal was to identify reasons for the runtime of crossover being a bottleneck in solving the generation expansion planning problem using linear programming. The second goal was to apply the knowledge obtained from tackling the first challenge to propose a faster solving approach. In this chapter, we provide a summary of how we attempted to tackle both challenges and we present the main findings and conclusions. Furthermore, in Section 9.1, we reflect upon the strengths and limitations of this research and provide suggestions for future work.

To investigate why crossover can be a bottleneck in the solving process, we solved several instances of the generation expansion planning problem using Gurobi. In one of these instances (see Figure 5.2b), the total runtime was dominated by the time spent in crossover. Symmetry, which was deliberately introduced in this problem instance, appeared to be the main cause for the longer crossover time. A comparison between solving the linear programming model with and without symmetry detection showed that symmetry detection and handling is beneficial for the runtime of the barrier method but detrimental to the runtime of crossover. For highly-symmetric instances, such as the one used for Figure 5.2b, the negative consequences for the crossover method outweigh the positive effects on the barrier method.

From these conclusions we determined that an ideal algorithm reduces symmetry so that barrier runs efficiently, but also unfolds the solution in a way which is more effective than applying crossover. Our approach for creating such an algorithm consists of using problem-specific knowledge on the generation expansion planning problem. We derived two types of symmetry that can occur in an instance of the generation expansion planning problem, one in which power plants within a node are symmetric and one in which power plants in different nodes are symmetric. For both types of symmetry, we showed which variables should be equal for symmetry to be present. Moreover, we distinguished between when the symmetry is a formulation symmetry and when the symmetry is not apparent from the formulation of the problem. For the former, we proved that the formulation symmetry is indeed present. For the latter, we explained why the problem can still be symmetric. The results throughout this thesis show that both types of symmetry have a detrimental effect on the runtime of crossover, while reducing the time spent by the barrier method.

Using the knowledge we obtained from deriving both types of symmetry, we came up with a novel approach to resolve each type of symmetry. This approach consists of three stages. First, we apply our knowledge of the symmetry to reduce the size of the initial problem instance. Then, we use a traditional linear programming solver to solve the reduced problem. Finally, we re-use the knowledge of the symmetry to retrieve a solution to the initial problem.

For the first type of symmetry, our method worked as follows. Given a pair of symmetric power plants within a country, we reduce the problem by merging both plants into a single plant with the same costs and capacities. After solving this reduced problem, we can obtain a solution to the initial problem by simply distributing the investment and production of the merged plant equally over the original plants.

Handling the second type of symmetry proved to be more difficult. A similar approach was taken. Instead of fully merging two symmetric plants in different nodes, we merged their production variables but kept their investment variables separate. Furthermore, we formulated additional constraints to account for the energy transmission between nodes and the production capacities, ensuring that the optimal solution to the reduced problem has costs equal to those of the initial problem. For the unfolding phase, we proposed a greedy algorithm which tries to fully fulfill the demand and net outgoing flow of one of two merged nodes, before assigning production to the other. This method can be used with many problem instances, as it only requires the variable production costs of two plants to be equal. Therefore, it does not only work for formulation symmetry, but can be applied to problem instances containing non-formulation symmetry as well.

For both symmetry resolving methods, we have proven that they return an optimal solution to the original problem. Furthermore, the results for the method resolving symmetry between power plants within a node showed that this method can be significantly faster than solely using a solver, in both the crossover phase and the total solving process. The second symmetry resolving algorithm, on the other hand, performs worse on average than only using Gurobi. However, some combinations of power plants to be merged led to a faster crossover runtime and total runtime. Therefore, depending on which plants are merged, our method could still be useful.

9.1. Discussion and Future Work

The results obtained in this research are not only relevant for quickly finding solutions to the generation expansion planning problem, the relevance of this thesis extends to crossover and linear programming in general. Our approach demonstrates that problem specific knowledge can be applied to reduce the runtime of crossover in highly symmetric problems. In the introduction we explained that crossover is often a computational bottleneck and therefore this thesis provides a useful addition to the recent developments regarding crossover.

In the progress of this research, Deakins, Knueven, and Ostrowski [42] also made an attempt in improving crossover for symmetric problems, which underpins the significance of developing alternatives to the traditional crossover method. They propose a method called orbital crossover, which tackles highly symmetric optimisation problems. Their method uses symmetries to aggregate a linear program and partition the variables and constraints. After solving the aggregated problem, the partition is used to convert the solution to the aggregated problem to an optimal vertex of the original linear program. Their results show that using orbital crossover is significantly faster than the crossover method of the open-source solver HiGHS on highly symmetric problems. On a less symmetric dataset the performance of orbital crossover was still better on average, but by a smaller margin and more problems were solved faster by HiGHS.

The similarities between their paper and our approach show that the principle of reducing the problem size using symmetry and applying information of this reduction to the crossover phase is an effective method when dealing with formulation symmetry. The difference between both algorithms is in how we reduce the problem and retrieve the final solution. Since our approach uses problem-specific knowledge, we are also able to reduce symmetries which are not apparent from the linear programming formulation. This can result in a larger total reduction of the problem. However, we cannot conclude that this benefits the efficiency of crossover, since our results show that only in a couple of cases our method is faster. The drawback of our method compared to the method of Deakins, Knueven, and Ostrowski [42] is that it requires this problem-specific knowledge, whereas their method is directly applicable to any linear programming problem. A suggestion for future work is to compare the effects of both methods.

In Chapter 2, we mentioned another paper providing a new crossover method, written by Ge, Wang, Xiong, *et al.* [22]. In this thesis, we did not compare our algorithm to their methods as we focused on resolving symmetry to speed up crossover, whereas they worked on alternatives for finding the optimal face of a problem and using this to obtain an optimal extreme point. In fact, both approaches could be employed together, so it could be useful to investigate the benefits of both methods and how they can be combined in future research.

Next to combining and comparing our method to other recently-proposed methods, we have several suggestions for improving our algorithms. Our method for dealing with symmetric power plants within a node could be modified so that it can also deal with different generation availability profiles. In that case, it does not rely on a formulation symmetry to be usable. However, as we have seen with our other algorithm, it could be that this does not result in a faster runtime.

For our second algorithm, we propose multiple improvements. Since it performs better for some selections of plants to be merged, it would be useful to research why these selections perform well. One possible reason could be that the merged plants in such a selection are employed in an optimal solution, since the merge then reduces the size of the optimal region of the search space. Furthermore, some selections can be larger than others due to the order in which they are compared.

Increasing the number of merged plants can also be done by merging multiple pairs of plants in two connected nodes. In Subsection 7.2.2, we discussed that this requires many additional constraints to account for merged plants producing more than the individual plants can produce or use. A useful direction for future work would be to identify which of all these constraints are actually required and if some of these constraints dominate other constraints. Another challenge that arises is finding a suitable unfolding algorithm. This is non-trivial since the assigned production of one pair of merged plants now influences the assignment for another pair of merged plants in the same merged node, since the demand has to be satisfied by the sum of all production values.

The same challenge arises when dealing with ramping. When ramping constraints are enforced, the unfolding algorithm should ensure that production values at consecutive time steps do not differ too much. Therefore, recovering the solution to the original problem is more difficult, as one cannot simply loop over all time steps and compute the production values at once.

Another improvement of our method could possibly be obtained by slightly relaxing some of the constraints we define for our reduced problem. The proofs we have written show that for each solution of the original problem there exists a solution to the reduced problem with the same objective value, and vice versa. However, we only require our algorithm to find an optimal solution to the original problem, so the reduced problem does not have to be completely equivalent to the original problem.

Lastly, we discuss a limitation of both symmetry resolving methods. This limitation is the lack of a proof that our method obtains a vertex of the solution space. Although our method always finds an optimal solution to the problem, crossover guarantees to find a vertex of the solution space and the corresponding basis. Algorithm 1 does not necessarily find a vertex as it computes investment values which do not necessarily lie at their upper or lower bound. We believe, however, that this algorithm can be easily modified to guarantee an optimal basic feasible solution, by computing the maximum possible investments and productions for each plant in a node one by one. Algorithm 2 probably finds a basic feasible solution as it produces as much as possible in the first of the two merged plants and produces the remaining in the other plant. However, we do not have a proof that this is definitely true, so we are not able to conclude this. For future work, it would be valuable to show that both algorithms actually find a basic feasible solution.

References

[1] N. E. Koltsaklis and A. S. Dagoumas, “State-of-the-art generation expansion planning: A review”, *Applied Energy*, vol. 230, pp. 563–589, 2018. DOI: `https://doi.org/10.1016/j.apenergy.2018.08.087`.

[2] E. Rothberg, *Parallelism in lp and mip*, 2020. [Online]. Available: `https://www.gurobi.com/wp-content/uploads/How-to-Exploit-Parallelism-in-Linear-and-Mixed-Integer-Programming.pdf`.

[3] P. Massé and R. Gibrat, “Application of linear programming to investments in the electric power industry”, *Management Science*, vol. 3, no. 2, pp. 149–166, 1957. DOI: `https://doi.org/10.1287/mnsc.3.2.149`.

[4] IEA, *World energy outlook 2022*, IEA Paris, France, 2022. [Online]. Available: `https://iea.blob.core.windows.net/assets/830fe099-5530-48f2-a7c1-11f35d510983/WorldEnergyOutlook2022.pdf`.

[5] V. Oree, S. Z. S. Hassen, and P. J. Fleming, “Generation expansion planning optimisation with renewable energy integration: A review”, *Renewable and Sustainable Energy Reviews*, vol. 69, pp. 790–803, 2017. DOI: `https://doi.org/10.1016/j.rser.2016.11.120`.

[6] G. A. Bakirtzis, P. N. Biskas, and V. Chatziathanasiou, “Generation expansion planning by milp considering mid-term scheduling decisions”, *Electric Power Systems Research*, vol. 86, pp. 98–112, 2012.

[7] S. Dehghan, N. Amjady, and A. Kazemi, “Two-stage robust generation expansion planning: A mixed integer linear programming model”, *IEEE Transactions on Power Systems*, vol. 29, no. 2, pp. 584–597, 2013. DOI: `https://doi.org/10.1109/TPWRS.2013.2287457`.

[8] A. Ramos, I. J. Perez-Arriaga, and J. Bogas, “A nonlinear programming approach to optimal static generation expansion planning”, *IEEE Transactions on Power Systems*, vol. 4, no. 3, pp. 1140–1146, 1989. DOI: `https://doi.org/10.1109/59.32610`.

[9] C. Antunes, A. Martins, and I. S. Brito, “A multiple objective mixed integer linear programming model for power generation expansion planning”, *Energy*, vol. 29, no. 4, pp. 613–627, 2004. DOI: `https://doi.org/10.1016/j.energy.2003.10.012`.

[10] J.-B. Park, Y.-M. Park, J.-R. Won, and K. Y. Lee, “An improved genetic algorithm for generation expansion planning”, *IEEE Transactions on Power Systems*, vol. 15, no. 3, pp. 916–922, 2000. DOI: `https://doi.org/10.1109/59.871713`.

[11] S. Kannan, S. Baskar, J. D. McCalley, and P. Murugan, “Application of nsga-ii algorithm to generation expansion planning”, *IEEE Transactions on Power systems*, vol. 24, no. 1, pp. 454–461, 2008. DOI: `https://doi.org/10.1109/TPWRS.2008.2004737`.

[12] R. J. Vanderbei *et al.*, *Linear programming*. Springer, 2020. DOI: `https://doi.org/10.1007/978-3-030-39415-8`.

[13] G. B. Dantzig, “Programming in a linear structure”, *Washington, DC*, 1948.

[14] E. Beale and R. Small, “Mixed integer programming by a branch and bound technique”, in *Proceedings of the IFIP Congress*, vol. 2, 1965, pp. 450–451.

[15] R. E. Bixby, “A brief history of linear and mixed-integer programming computation”, *Documenta Mathematica*, vol. 2012, pp. 107–121, 2012.

[16] I. J. Lustig, R. E. Marsten, and D. F. Shanno, “Interior point methods for linear programming: Computational state of the art”, *ORSA Journal on Computing*, vol. 6, no. 1, pp. 1–14, 1994. DOI: `https://doi.org/10.1287/ijoc.6.1.1`.

[17] G. B. Dantzig and M. N. Thapa, *Linear programming: Theory and extensions*. Springer, 2003, vol. 2. DOI: `https://doi.org/10.1007/b97283`.

[18] M. Bénichou, J.-M. Gauthier, P. Girodet, G. Hentges, G. Ribière, and O. Vincent, “Experiments in mixed-integer linear programming”, *Mathematical Programming*, vol. 1, pp. 76–94, 1971. DOI: `https://doi.org/10.1007/BF01584074`.

[19] R. E. Bixby, J. W. Gregory, I. J. Lustig, R. E. Marsten, and D. F. Shanno, "Very large-scale linear programming: A case study in combining interior point and simplex methods", *Operations Research*, vol. 40, no. 5, pp. 885–897, 1992. DOI: https://doi.org/10.1287/opre.40.5.885.

[20] E. Klotz and A. M. Newman, "Practical guidelines for solving difficult linear programs", *Surveys in Operations Research and Management Science*, vol. 18, no. 1-2, pp. 1–17, 2013. DOI: https://doi.org/10.1016/j.sorms.2012.12.001.

[21] L. Schork and J. Gondzio, "Implementation of an interior point method with basis preconditioning", *Mathematical Programming Computation*, vol. 12, no. 4, pp. 603–635, 2020. DOI: https://doi.org/10.1007/s12532-020-00181-8.

[22] D. Ge, C. Wang, Z. Xiong, and Y. Ye, "From an interior point to a corner point: Smart crossover", *arXiv preprint arXiv:2102.09420*, 2021. DOI: https://doi.org/10.48550/arXiv.2102.09420.

[23] E. D. Andersen and K. D. Andersen, "Presolving in linear programming", *Mathematical Programming*, vol. 71, pp. 221–245, 1995. DOI: https://doi.org/10.1007/BF01586000.

[24] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger, "Presolve reductions in mixed integer programming", *INFORMS Journal on Computing*, vol. 32, no. 2, pp. 473–506, 2020. DOI: https://doi.org/10.1287/ijoc.2018.0857.

[25] M. Kojima, S. Mizuno, and A. Yoshise, *A primal-dual interior point algorithm for linear programming*. Springer, 1989. DOI: https://doi.org/10.1007/978-1-4613-9617-8_2.

[26] K. Frisch, *The logarithmic potential method for convex programming, institute of economics, university of oslo, oslo, norway*, 1955.

[27] P. E. Gill, W. Murray, M. A. Saunders, J. A. Tomlin, and M. H. Wright, "On projected newton barrier methods for linear programming and an equivalence to karmarkar's projective method", *Mathematical programming*, vol. 36, no. 2, pp. 183–209, 1986. DOI: https://doi.org/10.1007/BF02592025.

[28] C. Maes, Z. Gu, E. Rothberg, and R. Bixby, *Initial basis selection for lp crossover*, 2014. [Online]. Available: https://cerfacs.fr/wp-content/uploads/2016/01/maes.pdf.

[29] E. Panos, *Iea-etsap | energy systems analysis*, Mar. 2022. [Online]. Available: https://iea-etsap.org/webinar/CPLEX%20options%20for%20running%20TIMES%20models.pdf.

[30] R. E. Bixby and M. J. Saltzman, "Recovering an optimal lp basis from an interior point solution", *Operations Research Letters*, vol. 15, no. 4, pp. 169–178, 1994. DOI: https://doi.org/10.1016/0167-6377(94)90074-4.

[31] N. Megiddo, "On finding primal-and dual-optimal bases", *ORSA Journal on Computing*, vol. 3, no. 1, pp. 63–65, 1991. DOI: https://doi.org/10.1287/ijoc.3.1.63.

[32] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2023. [Online]. Available: https://www.gurobi.com.

[33] M. Ruthmaier, Forum Post. [Online]. Available: https://support.gurobi.com/hc/en-us/community/posts/17441473898641/comments/17819937029521.

[34] IBM, *Cplex*. [Online]. Available: https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer.

[35] M. E. Pfetsch and T. Rehn, "A computational comparison of symmetry handling methods for mixed integer programs", *Mathematical Programming Computation*, vol. 11, pp. 37–93, 2019. DOI: https://doi.org/10.1007/s12532-018-0140-y.

[36] F. Margot, "Symmetry in integer linear programming", *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pp. 647–686, 2009. DOI: https://doi.org/10.1007/978-3-540-68279-0_17.

[37] B. D. McKay and A. Piperno, "Practical graph isomorphism, ii", *Journal of Symbolic Computation*, vol. 60, pp. 94–112, 2014, ISSN: 0747-7171. DOI: https://doi.org/10.1016/j.jsc.2013.09.003.

[38] *Saucy*. [Online]. Available: http://vlsicad.eecs.umich.edu/BK/SAUCY/.

[39] TCS, *Bliss*. [Online]. Available: http://www.tcs.hut.fi/Software/bliss/index.html.

[40] D. Salvagnin, "A dominance procedure for integer programming", *Unpublished master's thesis, University of Padova, Padova, Italy*, 2005.

[41] K. Bestuzheva, M. Besançon, W.-K. Chen, *et al.*, "The scip optimization suite 8.0", *arXiv preprint arXiv:2112.08872*, 2021. DOI: https://doi.org/10.48550/arXiv.2112.08872.

[42] E. Deakins, B. Knueven, and J. Ostrowski, "Orbital crossover", 2023.

A

Formulation and Proof of Symmetry Reduction on Symmetric Power Plants in Connected Nodes

A.1. Linear Program for the Reduced Problem

$$\text{minimize} \quad \sum_{\{g\in\mathcal{G}\}} IC_g \cdot UCAP_g \cdot ui_g + WOP \cdot \left(\sum_{\{g\in\mathcal{G}',t\in\mathcal{T}\}} (PC_g \cdot p_{g,t}) + \sum_{\{n\in\mathcal{N},t\in\mathcal{T}\}} (MDC \cdot md_{n,t}) \right) \tag{A.1a}$$

subject to

$$p_{g,t} \leq \sum_{g_{sub}\in SUB_g} GA_{g_{sub},t} \cdot UCAP_{g_{sub}} \cdot ui_{g_{sub}} \qquad \forall g \in \mathcal{G}', t \in \mathcal{T} \tag{A.1b}$$

$$\begin{aligned} &\sum_{g\in\{g'\in\mathcal{G}'|PN_{g'}=n\}} p_{g,t} + \sum_{l\in\{l'\in\mathcal{L}|NB_{l'}\in SUBN_n\}} f_{l,t} - \sum_{l\in\{l'\in\mathcal{L}|NA_{l'}\in SUBN_n\}} f_{l,t} \\ &+ \sum_{n_{sub}\in SUBN_n} md_{n_{sub},t} = \sum_{n_{sub}\in SUBN_n} D_{n_{sub},t} \qquad \forall n \in \mathcal{N}', t \in \mathcal{T} \end{aligned} \tag{A.1c}$$

$$f_{l,t} \geq -ICAP_l \qquad \forall l \in \mathcal{L}, t \in \mathcal{T} \tag{A.1d}$$

$$f_{l,t} \leq ECAP_l \qquad \forall l \in \mathcal{L}, t \in \mathcal{T} \tag{A.1e}$$

$$\begin{aligned} &\sum_{g\in\{g'\in\mathcal{G}'|PN_{SUB_{g'}}=\{SUBN_{n,1}\}\}} p_{g,t} + \sum_{l\in\{l'\in\mathcal{L}|NB_{l'}=SUBN_{n,1}\}} f_{l,t} \\ &- \sum_{l\in\{l'\in\mathcal{L}|NA_{l'}=SUBN_{n,1}\}} f_{l,t} + md_{SUBN_{n,1},t} \leq D_{SUBN_{n,1}} \qquad \forall n \in \mathcal{N}_{new}, \forall t \in \mathcal{T} \end{aligned} \tag{A.1f}$$

$$\begin{aligned} &\sum_{g\in\{g'\in\mathcal{G}'|PN_{SUB_{g'}}=\{SUBN_{n,2}\}\}} p_{g,t} + \sum_{l\in\{l'\in\mathcal{L}|NB_{l'}=SUBN_{n,2}\}} f_{l,t} \\ &- \sum_{l\in\{l'\in\mathcal{L}|NA_{l'}=SUBN_{n,2}\}} f_{l,t} + md_{SUBN_{n,2},t} \leq D_{SUBN_{n,2}} \qquad \forall n \in \mathcal{N}_{new}, \forall t \in \mathcal{T} \end{aligned} \tag{A.1g}$$

$$\begin{aligned} &p_{g_{new},t} + \sum_{g\in\{g'\in\mathcal{G}'|PN_{SUB_{g'}}=\{PN_{SUB_{g_{new}},1}\}\}} p_{g,t} + \sum_{l\in\{l'\in\mathcal{L}|NB_{l'}=SUBN_{n,1}\}} f_{l,t} \\ &- \sum_{l\in\{l'\in\mathcal{L}|NA_{l'}=SUBN_{n,1}\}} f_{l,t} + md_{SUBN_{n,1},t} \end{aligned}$$

$$\leq D_{SUBN_{n,1}} + GA_{SUB_{g_{new},2},t} \cdot UCAP_{SUB_{g_{new},2}} \cdot ui_{SUB_{g_{new},2}} \qquad \forall g_{new} \in \mathcal{G}_{new}, \forall t \in \mathcal{T} \tag{A.1h}$$

$$p_{g_{new},t} + \sum_{g \in \{g' \in \mathcal{G}' | PN_{SUB_{g'}} = \{PN_{SUB_{g_{new},2}}\}\}} p_{g,t} + \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = SUBN_{n,2}\}} f_{l,t}$$
$$- \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = SUBN_{n,2}\}} f_{l,t} + md_{SUBN_{n,2},t}$$
$$\leq D_{SUBN_{n,2}} + GA_{SUB_{g_{new},1},t} \cdot UCAP_{SUB_{g_{new},1}} \cdot ui_{SUB_{g_{new},1}} \qquad \forall g_{new} \in \mathcal{G}_{new}, \forall t \in \mathcal{T} \tag{A.1i}$$

$$p_{g,t} \geq 0 \qquad \forall g \in \mathcal{G}', t \in \mathcal{T} \tag{A.1j}$$

$$md_{n,t} \geq 0 \qquad \forall n \in \mathcal{N}, t \in \mathcal{T} \tag{A.1k}$$

$$md_{n,t} \leq D_{n,t} \qquad \forall n \in \mathcal{N}, t \in \mathcal{T} \tag{A.1l}$$

$$ui_g \geq 0 \qquad \forall g \in \mathcal{G} \tag{A.1m}$$

A.2. Proof

To prove that the described algorithm is effective for reducing symmetries between power plants in connected, we need to show two things. First, we show that given an optimal solution to (3.1), there exists a feasible solution to (A.1) with the same optimal value. Second, we show that given an optimal solution to (A.1), Algorithm 2 finds a feasible solution to (3.1) with the same objective value. If both conditions hold, this means that the optimal objective values of both problems are the same and, therefore, that our method computes an optimal solution to (3.1).

$\Rightarrow$

For the first part of the proof, let us assume that we have an optimal solution to an arbitrary instance of (3.1) without the integrality constraint of (3.1l) and the ramping constraints (3.1f) and (3.1g).

From this solution, we can construct the following solution to (A.1). All variables except the production values of merged nodes will be equal to the corresponding variable in the solution to (3.1). For an arbitrary merged plant g_{new} originating from g_1 in n_1 and g_2 in n_2, we will set its production value $p_{g_{new},t}$ equal to the sum of $p_{g_1,t}$ and $p_{g_2,t}$.

We first show that this solution is in the feasible region of (A.1), i.e. all constraints of this problem are satisfied. For constraints (A.1d), (A.1e), (A.1k), (A.1l) and (A.1m), this is trivial since the variables remain unchanged and the constraints were already present in (3.1). For (A.1b) and (A.1j) the same holds if g is not a merged node. If $g \in \mathcal{G}_{new}$, we can show that (A.1b) is satisfied by adding (3.1b) for both subplants of g and (A.1j) follows from the fact that $p_{g,t}$ is the sum of two nonnegative numbers.

To show that (A.1c) is satisfied, we consider two cases. If $n \in \mathcal{N}'$ contains no merged power plants, (A.1c) is exactly the same as (3.1c). If $n \in \mathcal{N}'$ does contain a merged power plant and, therefore, has two subnodes n_1 and n_2, we derive the following equality using (3.1c) for all t:

$$\sum_{g \in \{g' \in \mathcal{G}' | PN_{g'} = n\}} p_{g,t} + \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} \in SUBN_n\}} f_{l,t} - \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} SUBN_n\}} f_{l,t} + \sum_{n_{sub} \in SUBN_n} md_{n_{sub},t} =$$
$$\sum_{g \in \{g' \in \mathcal{G} | PN_{g'} = n_1\}} p_{g,t} + \sum_{g \in \{g' \in \mathcal{G} | PN_{g'} = n_2\}} p_{g,t} + \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = n_1\}} f_{l,t} - \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = n_1\}} f_{l,t} \tag{A.2}$$
$$+ \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = n_2\}} f_{l,t} - \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = n_2\}} f_{l,t} + md_{n_2,t} + md_{n_2,t} = D_{n_1,t} + D_{n_2,t} = \sum_{n_{sub} \in SUBN_n} D_{n_{sub},t}.$$

The satisfaction of (A.1f) follows directly from (3.1c), because for a node $n \in \mathcal{N}_{new}$ with subnodes n_1 and n_2

$$\sum_{g \in \{g' \in \mathcal{G}' | PN_{SUB_{g'}} = \{SUBN_{n,1}\}\}} p_{g,t} \leq \sum_{g \in \{g' \in \mathcal{G} | PN_{g'} = n_1\}} p_{g,t}. \tag{A.3}$$

This holds since the left-hand side is equal to the production of all non-merged plants located in n_1, whereas the right-hand side also contains part of the production of the merged plant of n. The same principle can be applied to n_2 to show satisfaction of (A.1g).

The remaining two sets of constraints are (A.1h) and (A.1i). As they are equivalent apart from the node of the original problem used, we only show satisfaction for one of the two. The other can be shown using the exact same procedure. To show satisfaction of (A.1h) for an arbitrary plant g_{new} with subplants g_1 and g_2, we add and subtract $p_{g_1,t}$ to the left-hand side and apply (3.1c) to obtain

$$\begin{aligned} &p_{g_{new},t} + p_{g_1,t} - p_{g_1,t} + \sum_{g \in \{g' \in \mathcal{G}' | PN_{SUB_{g'}} = \{n_1\}\}} p_{g,t} + \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = n_1\}} f_{l,t} - \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = n_1\}} f_{l,t} + md_{n_1,t} = \\ &p_{g_{new},t} - p_{g_1,t} + D_{n_1,t} = p_{g_2,t} + D_{n_2,t}. \end{aligned} \tag{A.4}$$

Now, we can use (3.1b), which states that $p_{g_2,t}$ is smaller than or equal to its production capacity, to derive the satisfaction of (A.1h).

We, finally, show that the constructed solution also has the same objective value as the solution to the original problem. Consider the objective value of the constructed solution given by (A.1a). We rewrite each of the production values $p_{g,t}$ to obtain

$$\sum_{g \in \mathcal{G}} IC_g \cdot UCAP_g \cdot ui_g + WOP \cdot \left(\sum_{g \in \mathcal{G}'} \sum_{g_{sub} \in SUB_g, t \in \mathcal{T}} (PC_{g_{sub}} \cdot p_{g_{sub},t}) + \sum_{n \in \mathcal{N}, t \in \mathcal{T}} (MDC \cdot md_{n,t}) \right). \tag{A.5}$$

Since all sub-plants of plants in $\mathcal{G}'$ together form the set $\mathcal{G}$, the double summation over the generators can be combined into one sum over $\mathcal{G}$, giving us an objective value equal to (3.1a).

$\Leftarrow$

For the reverse implication, suppose we have an optimal solution to an arbitrary instance of (A.1) without the integrality and ramping constraints. Without loss of generality we assume that it contains exactly one merged node n_{new}, originating from n_1 and n_2 and that its merged plant g_{new} has subplants g_1 and g_2. If multiple pairs of nodes are merged, the exact same procedure can namely be used to prove this condition. We show that Algorithm 2 produces a feasible solution to (3.1) with the same objective value.

For all constraints of (3.1) not involving $p_{g_1,t}$ or $p_{g_2,t}$, it follows directly from (A.1) that they are satisfied, since our unfolding algorithm does not alter those variables and the constraints of (A.1) have the exact same effects on these variables as the constraints of (3.1).

It remains to show that (3.1b) and (3.1h) are satisfied for both g_1 and g_2, and that (3.1c) is satisfied for both n_1 and n_2. To show this, let us first recall (7.20) and (7.21). These equations state that

$$p_{g_1,t} = D_{n_1,t} - md_{n_1,t} + \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = n_1\}} f_{l,t} - \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = n_1\}} f_{l,t} - \sum_{g \in \{g' \in \mathcal{G}' | PN_{SUB_{g'}} = \{PN_{SUB_{g,1}}\}\}} p_{g,t},$$

and $p_{g_2,t} = p_{g_{new},t} - p_{g_1,t}$, respectively.

To prove satisfaction of (3.1b) for g_1, we use (A.1c) to obtain that $p_{g_1,t}$ is equal to

$$\begin{aligned} &p_{g_{new},t} + \sum_{g \in \{g' \in \mathcal{G}' | PN_{SUB_{g'}} = \{SUBN_{n,2}\}\}} p_{g,t} + \sum_{l \in \{l' \in \mathcal{L} | NB_{l'} = SUBN_{n,2}\}} f_{l,t} \\ &- \sum_{l \in \{l' \in \mathcal{L} | NA_{l'} = SUBN_{n,2}\}} f_{l,t} + md_{SUBN_{n,2},t} - D_{SUBN_{n,2}}, \end{aligned} \tag{A.6}$$

which according to (A.1i) is smaller than or equal to the production capacity of g_1. For g_2, we can show this using (A.1h), as $p_{g_{new},t} - p_{g_1,t}$ is equal to the left-hand side of that constraint minus D_{n_1}. Therefore, it immediately follows that $p_{g_2,t}$ is smaller than the production capacity of g_2.

To show that (3.1h) is satisfied for g_1, we note that $p_{g_1,t}$ is equal to the right-hand side of (A.1f) minus the left-hand side of that inequality. Thus, $p_{g_1,t} \geq 0$. For g_2, we apply (3.1c) to derive that

$$\begin{aligned} p_{g_{new},t} - p_{g_1,t} = D_{n_2,t} - &\sum_{g \in \{g' \in \mathcal{G}' \mid PN_{SUB_{g'}} = \{SUBN_{n,2}\}\}} p_{g,t} + \sum_{l \in \{l' \in \mathcal{L} \mid NB_{l'} = SUBN_{n,2}\}} f_{l,t} \\ - &\sum_{l \in \{l' \in \mathcal{L} \mid NA_{l'} = SUBN_{n,2}\}} f_{l,t} + md_{SUBN_{n,2},t}. \end{aligned} \tag{A.7}$$

This is equal to the right-hand side of (A.1g) minus its right-hand side, so $p_{g_2,t} \geq 0$.

Finally, we need to show that (3.1c) is satisfied for both n_1 and n_2. For n_1, this is trivial due to the definition of $p_{g_1,t}$. We can namely move all variables except $D_{n_1,t}$ to the left-hand side to obtain (3.1c). Now, note that it immediately follows that (3.1c) is satisfied for n_2, because we can subtract both sides of (3.1c) for g_1 from the corresponding sides of (A.1c) to obtain (3.1c) for g_2.

Now that we have shown that Algorithm 2 produces a feasible solution to (3.1), it only remains to be shown that this solution has the same objective value as the solution to (A.1). To do this, consider the objective value of the constructed solution by Algorithm 2. We rewrite its objective value (3.1a) to separate g_1 and g_2. This gives us an objective value of

$$\begin{aligned} &\sum_{g \in \mathcal{G}} IC_g \cdot UCAP_g \cdot ui_g + WOP \cdot \Bigg(PC_{g_1} \cdot p_{g_1,t} + PC_{g_2} \cdot p_{g_2,t} \\ &+ \sum_{g \in \mathcal{G} \setminus \{g_1, g_2\}, t \in \mathcal{T}} (PC_g \cdot p_{g,t}) + \sum_{n \in \mathcal{N}, t \in \mathcal{T}} (MDC \cdot md_{n,t}) \Bigg). \end{aligned} \tag{A.8}$$

Since, PC_{g_1} is equal to PC_{g_2} and to $PC_{g_{new}}$, we can rewrite this equation to obtain

$$\sum_{g \in \mathcal{G}} IC_g \cdot UCAP_g \cdot ui_g + WOP \cdot \Bigg(PC_{g_{new}} \cdot p_{g_{new},t} + \sum_{g \in \mathcal{G}' \setminus \{g_{new}\}, t \in \mathcal{T}} (PC_g \cdot p_{g,t}) + \sum_{n \in \mathcal{N}, t \in \mathcal{T}} (MDC \cdot md_{n,t}) \Bigg), \tag{A.9}$$

which is equal to (A.1a). □