Designing Multi-Energy Systems

Noortje Bonenkamp

Designing Multi-Energy Systems

Solution methods for a multi-period network design problem

by

Noortje Bonenkamp

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Thursday July 9, 2020 at 10:00 AM.

This research was undertaken in partial fulfillment of the Master's Applied Mathematics, with an annotation in Technology in Sustainable Development (TiSD). ORTEC B.V. has agreed to supervise and support this research, in addition to supervision from TU Delft.

Student number:	4323769	
Date:	July 3, 2020	
Thesis committee:	Dr. ir. J.T. van Essen	TU Delft
	Dr. D.C. Gijswijt	TU Delft
	Dr. ir. M.B. van Gijzen	TU Delft
	I. van Beuzekom, MSc	ORTEC

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft ORTEC

Preface

The work in front of you is the result of my thesis for a MSc title in Applied Mathematics at Delft University of Technology. I would like to thank several people that made it possible for me to finish this work.

I was lucky to have two very dedicated supervisors during this whole process: Iris and Theresia. Thank you, Iris, for your endless enthusiasm, ideas and for always reminding me of the bigger picture. Thank you Theresia, for the motivating meetings, all your advice and for lots and lots of proofreading. It was great to work with you both.

I look back at a wonderful internship experience at ORTEC. I would like to thank my colleagues for interesting talks and for a lot of fun in the office. Special thanks to Bolor and Tim for help with coding in Pyomo, to the *linksachterin* crew for all the songs on Friday, and to Julie for many great talks.

A large part of this thesis is written from home, both in Delft and at my parents place. To my roommates in Delft: thank you for all the support and comic relief. Especially Bas, for much help and coffee, and Charlotte and Merlijn: it was a pleasure to be in the same graduation boat. And lastly, to my parents and my brothers: thank you for a great time during our lockdown together. You are my favorite network!

Noortje Bonenkamp
Delft, July 2020

Nomenclature

Parameters

η^k Standing loss factor for storage of energy carrier k

μ_{ij} Multiplicator on arc/edge (i,j) representing distance losses or conversion rates

ξ^k Efficiency loss factor for storage of energy carrier k

b_{it} Demand at node i at time t

c_t^k Costs of building a single storage unit for carrier k at time t

c_t^s Costs of building a single supply unit s at time t

c_{ijt} Costs of building a single asset on arc/edge (i,j) at time t

g_{it} Gas supply at node i at time t

u^k Capacity of a single storage unit for energy carrier k

u_t^s Capacity of a supply unit s at time t

u_{ij} Capacity of a single asset on arc/edge (i,j)

Sets

E Set of edges

K Set of energy carriers

L Set of locations

T Set of time periods

V Set of nodes

Variables

W_{it}^{IN} Energy flow that gets stored at location i at time t

W_{it}^{OUT} Energy flow that gets supplied from storage at location i at time t

x_{ijt} Energy flow on edge (i, j) at time t

z_{ijt} Number of assets built at arc/edge (i, j) at time t

z_{it}^{S} Number of electricity supply units of type s built at node i at time t

z_{it}^{W} Number of storage units built at node i at time t

Glossary

CHP combined heat- and power plant.

GHG greenhouse gases.

HP heat pump.

MES multi-energy system.

P2G power-to-gas unit.

RES renewable energy sources.

Contents

1

Introduction

In this introduction, we first explain the motivation for this research by introducing the urgency for and challenge of designing multi-energy systems. Second, we introduce the research objective and questions, and provide the outline of this report. Third, we explain the scientific and practical relevance of this research.

1.1. Research motivation

Climate change is one of the biggest challenges of the 21st century. In order to prevent the world from its worst impacts, the transition to a low-carbon society has to be made as quickly as possible. This is defined as the Energy Transition, which aims to transform the current energy system that is based on fossil fuels into one that is based on renewable energy sources. Creating a society that can function without fossil fuels, while maintaining maintaining or even increasing the current level of prosperity is a highly complex process in which decision makers have to consider multiple environmental, social and financial objectives.

Transforming the energy system

The energy supply sector is the largest contributor to global emissions from greenhouse gases (GHG) (Bruckner et al., 2014). Under the Paris agreement, world-wide targets on reducing GHG emissions and increasing the share of renewable energy sources (RES) have been set, pushing the decarbonization of our energy system (European Commission, 2015). Multiple options exist for improving the design of energy systems, such as energy efficiency improvements, fugitive emission reduction, implementation of low GHG energy supply technologies, such as RES and nuclear power, and carbon capture- and storage. Due to significant efforts worldwide at various levels, RES have been increasingly implemented in the power generation mix over the last years (Bruckner et al., 2014). However, two challenges arise here. First, decarbonization of existing uses of electricity is only part of the story. Other energy sectors, such as heating and cooling, are also a major contribution to energy consumption as well as GHG emissions, and these sectors are harder to decarbonize than electricity (Mancarella, 2014). Second, RES can have large fluctuations in supply at different temporal scales and provide mostly electricity, whereas that is currently only less than 20% of the worldwide energy demand (IEA, 2019).

Future energy systems should be able to cope with these *mismatches* in both energy carrier and time. To achieve a truly reliable and sustainable energy system, long-term strategies that address *all* energy sectors are required.

Integrating multiple energy carriers (for example electricity, heating/cooling and gas) in a holistic whole-energy system could address these challenges (Mancarella, 2014). Such a coupled system has the potential to provide secure and affordable energy to present and future generations while meeting ambitious environmental targets. Better understanding and development of these so-called *integrated* or *multi-energy system (MES)*s, where different energy carriers optimally interact at different levels, is therefore crucial for the energy systems of the future. In MES, the interactions between energy carriers are exploited in the design and operation phase, improving the technical, environmental end economic performance of the system. Combining or coupling the traditionally separate energy systems can result in a number of benefits, as it becomes possible to take advantage of specific virtues of each energy carrier (Geidl et al., 2007). For example, electricity can easily be produced in a sustainable way, but is inefficient and expensive to store, especially at mid to long-term; whereas a natural gas equivalent is difficult to produce in a sustainable way, but can be stored using relatively simple and cheap technologies. Through converter devices, that transform one energy carrier into another, separate energy systems can be coupled, which enables exchange of energy among them.

Figure 1.1: Integrating different energy infrastructures is crucial for energy systems of the future. *Source*: Kopernikus/BMBF.

Optimizing multi-energy systems

Due to the high variety of possibilities both in how energy is used and in how it is converted and/or stored, optimizing the design and operation of a multi-energy system results in a complex problem with many decision variables. With the increasing interaction of the electrical smart grid with other energy sectors, it becomes more important to develop models that are

capable of dealing with the full complexity of MES problems to support policy makers and industry. This becomes even more apparent when we consider the long-term consequences and high impact of decisions on energy infrastructure. Various models and methods have been developed to find the optimal coupling and energy exchange among multiple energy carriers based on various criteria such as cost, emissions, energy efficiency, availability, security, and other parameters (Geidl et al., 2007). However, most existing models are not capable of optimizing both geographical and temporal aspects of all energy carriers (Van Beuzekom et al., 2016), which therefore represents a key research opportunity.

In this research, we investigate a model that intends to optimize the design of a MES in which electricity, gas and heat networks for mid-size cities are combined through conversion and storage assets. The developed model builds upon research of Van Beuzekom (2017), which has a unique approach in addressing a MES that combines e.g. temporal and geographical constraints for the investment planning of three energy carriers. The model intends to optimize the topology and operation of electricity, heat and gas networks simultaneously, given demand and resources. Solving real-case problem instances of this model with state-of-the-art solvers is computationally challenging. In this research, we intend to examine computational "bottlenecks" of this model, and provide strategies to reduce the resulting optimality gap.

1.2. Research objective and questions

This research aims to solve the proposed multi-energy systems model in an efficient way. Our main research question is stated as follows.

What are good solution methods for optimizing the design of multi-energy systems, using the developed model?

With the following sub-questions:

1. *What are suitable solution methods for solving these types of problems?*
2. *How difficult is the model from both a theoretical and practical viewpoint?*
3. *How succesful are the solution methods for solving large instances and can these problem instances be solved to (near) optimality?*
4. *What is the influence of taking into account Brownfield data on the complexity of the model?*

Report outline

This report is structured around the subquestions. After introducing the concept of multi-energy systems in more depth, as well as the fundamentals of network design and relevant previous work in Chapter 2, we intend to answer subquestion 1. In Chapter 3, we introduce details of the model and our formulation. In addition, we analyze the model's computational complexity and characteristics within a network optimization framework, and intend to answer

the first part of subquestion 2. In Chapter 4, we intend to answer the second part of this subquestion, by examining the performance of a state-of-the-art solver and, subsequently, investigate how its settings can be improved for solving a set of varying problem instances. In Chapter 5, we formulate various valid inequalities for our model that can be used as an extension to the standard solver. In Chapter 6, we introduce Benders- and Lagrangian based algorithms for our model. In Chapter 7, we perform various computational experiments and intend to answer subquestions 3 and 4. With our findings, we can answer the main question and provide ideas for future research in Chapter 8.

Research scope

The model can be formulated in different ways, but we only investigate and conduct experiments with a single formulation that we base on previous work. Even though the long-term planning of energy infrastructure consists of many uncertain factors both on the supply- and demand side, we limit this research to deterministic optimization. In addition, our focus lies mainly on mathematical programming methods for reasons that will become apparent in the following chapters.

1.3. Relevance

This study has both scientific and practical relevance.

Scientific relevance

The proposed model by Van Beuzekom (2017) is the first MES framework combining spatial and long-term temporal aspects for three energy networks, and therefore, contributes to the existing body of MES research. Investigating this model in a mathematical optimization context, we are dealing with a strongly NP-hard network design problem with many additional complicating factors, as will become apparent in the following chapters. To the best of our knowledge, there have not been many studies of network optimization models that share all of these complicating factors. For instance, we consider a purely integer objective function, as operational costs are modeled as energy losses. In addition, we are dealing with generalized flow constraints due to these energy losses and conversion rates. Another complicating factor is the possibility of flow to travel to the next time period through storage assets. Network optimization models consisting of all of these factors have not yet been investigated. In addition, very few studies on (multi-period) network design problems that model energy infrastructure have been conducted.

Practical relevance

The goal of the proposed model is to create a framework that helps decision makers on different levels with designing a pathway to a sustainable and reliable energy system in the most cost-efficient manner. Though there are many uncertainties in any long-term planning situation, especially one with so many variables as an energy transition, the model can be insightful

for analyzing difficult trade-offs between investment possibilities. This is especially interesting in the field of MES, as there are no fully coupled models that can provide this insight.

Although the model that we investigate can be applied to various scales, we will demonstrate its use on a city-scale in this research. Cities play a crucial role in the battle against climate change. As cities currently emit more than 70% of the GHG emissions and an increasing proportion of the world's population is moving to urban areas, improving the design of urban energy systems is crucial for reaching environmental targets (Samsatli and Samsatli, 2018). At the same time, cities are at high risk from climate change consequences. Determined to take action, many cities have been shown to committing themselves to challenging sustainable development goals (Van Beuzekom et al., 2016). A multi-energy perspective on the urban energy system is crucial for achieving these goals, which is why we foresee great benefit from this framework for urban decision makers. Providing decision makers with a good model might provide them with insights that incentivize them to take action on different levels in the - for them often rather unknown - field of MES.

By investigating suitable mathematical optimization methods for solving real-case problem instances in this research, we intend to improve computing times for finding good solutions. On a city level, even a small improvement in the objective value corresponds to large costs savings in infrastructure investments. Therefore, improving the model's objective and computing times has a significant effect from an economic point of view. In addition, the model can support decision makers with achieving a more reliable and sustainable energy system, as it takes environmental and demand constraints over the whole energy system into account, while optimizing over all of these simultaneously. Improving ways in which the general model is solved considering all of these constraints is attractive from a social- and environmental point of view. More specifically, as we intend to understand and improve the most general version of the model, modifications or extensions to the model might be easier to make, which can make the model more useful and therefore more attractive to apply in practical cases.

2

Background

In this chapter, we first discuss previous work on multi-energy systems and introduce the model that our research builds upon. Second, we review the fundamentals of network design and discuss solution methods that are relevant to our model. Third, we discuss the theory of computational complexity.

2.1. Multi-energy systens

In the modern world, different forms of energy are required, such as coal, petroleum products, biomass, and grid-bound energy carriers like electricity, natural gas, and district heating/cooling. Different energy forms are managed and transported through their own infrastructure. Interactions between these energy infrastructures have always taken place at different levels and are increasing (Mancarella, 2014): from a physical perspective, all energy systems are "multi-energy". However, the different energy infrastructures are considered and operated almost independently.

The need for a whole-system perspective

The concept of multi-energy systems refers to considering a holistic system-perspective in optimization and evaluation of a case (for example within a building, a city or an entire country). Because there is a need for decarbonization of our *entire* energy system, and not just electricity, the role of energy systems integration has recently increased in importance (Mancarella et al., 2016). In classical case studies, often only one energy sector is considered (such as heat or electricity), whereas the approach for analyzing MES expands the boundary beyond individual sectors, taking a whole-system perspective on all of the activities in the energy system. This perspective allows for identifying how traditionally independent energy systems can best be integrated to improve their collective performance. The question now arises as to what is the optimal way of integrating our energy systems.

There is a growing body of literature that recognises the importance of multi-energy systems. Geidl et al. (2007) were one of the first to propose the integration of energy systems as an important concept for future energy systems. Introducing the concept of *energy hubs*, i.e., units where multiple energy carriers can be converted, conditioned and stored, they inspired much

follow-up research in the field of MES. Kroposki and OMalley (2012) introduce different examples of successful energy system integration. Mancarella et al. (2014) provide a, now rather outdated, overview of different models and assessment techniques that are available to analyze MES. They stated that, although it is becoming more established that MES perform better than classical separate energy systems from energy, environmental, and techno-economic perspectives, there is a lack of a comprehensive view on them. A single definition of a MES has not yet been made, which is why it is challenging for researchers and policy makers to find common terminology and system boundaries.

Modeling MES

In another paper, Mancarella et al. (2016) addressed the need for modeling the (multi) energy system as a whole, instead of separate energy sectors. Modeling helps to assess interactions between different energy sectors and get insight into benefits and potential (or unforeseen) constraints and drawbacks from systems integration. They address different challenges. First, they highlight the often fragmented nature of ownership and operational responsibilities in energy systems, despite their physical interaction. Responsibilities are split up among a number of parties, that ideally compete with each other. The natural monopolies that often define energy network infrastructures further complicate interaction between different actors in MES. One can think of examples in which options have moderate benefits in separate energy systems, but which can be very attractive for the energy system as a whole. Currently, the incentives for industry to invest in such solutions might not be strong enough. Second, misalignment of the regulation and financing of different networks can be an issue. For instance, heat networks are often treated as private investments, while electricity- and gas networks are heavily regulated and benefit from privileged conditions or financing. If “business as usual” regulations do not transform our energy-systems into a MES, then there might be a need to revise the set of responsibilities and opportunities that different actors, such as owners, operators, energy retailers, etc., have and to introduce new regulations on existing parties. These changes cannot be made without evidence that they will have the desired effect. This again points out the need for accurate MES models.

Unfortunately, modeling MES in an accurate way is a complex task. First and foremost because of the added complexity of combining multiple infrastructures. In the majority of studies on MES, the problem is therefore often simplified beforehand, by analyzing or optimizing the systems separately and connecting them iteratively (Van Beuzekom et al., 2020). The question of what level of detail is needed is challenging. This is often the case for modeling complex systems, but even more so for modeling MES, since different levels of simplifications/reductions can be required for different sub-systems. In addition, it is challenging to adopt suitable temporal- and spatial scales and perspectives as different scales should be considered for different energy sectors (Mancarella et al., 2016). For instance, for an electricity and heat network, different time periods are interesting to use. Particularly sensitive to the choice of timescales is the modeling of energy storage, which is something that becomes increasingly important (Ruth and Kroposki, 2014). However, most existing MES models are not

capable of optimizing the spatial and temporal aspects of all energy carriers.

The use of mixed-integer linear programming (MILP) for modeling energy optimization problems has proven to be a particularly useful tool in literature (Gabrielli et al., 2018). MILP can capture the features of energy problems well and approximate solutions within reasonable computation times; although this often requires significant model simplifications to deal with the complexity of such problems. There are many studies in which objectives for individual energy networks at a variety of scales are optimized using MILP (Adams and Laughton, 1974; Chang, 2014; Hugo et al., 2005). Less publications focus on the integrated flow of combined energy networks. In most of these, a network of two energy sectors is modeled using MILP, as opposed to the three energy sectors that we consider in this thesis (Casisi et al., 2009; Gabrielli et al., 2018; Wakui et al., 2014; Weber and Shah, 2011; Wouters et al., 2015). In addition, in most of these publications a static case is considered. There are examples of multi-period optimization problems, but they mostly consider a time period of maximally one year (Omu et al., 2013; Weber and Shah, 2011), using a number of design days. Besides that, often the operations of a network were optimized, instead of the network design or the (long-term) investment strategy. Longer time horizons for investigating MES have not yet been considered at the design phase (Gabrielli et al., 2018). However, it is crucial to consider long-term investments on infrastructure design: not only to enable an energy transition, incorporate large amounts of RES, and reach the stringent 2050 climate goals, but also because, more practically, optimizing the design of energy networks can provide significant cost savings in the order of hundreds of billions of dollars (Balakrishnan et al., 2017).

A long-term MES planning model

Van Beuzekom et al. (2017) were the first to propose an optimal long-term investment planning model for energy networks in medium-sized cities which couples long-term planning of all three main energy infrastructures: electricity, gas and heat. In the following chapter, we formally introduce the concepts of the model and our formulation of it. The model's approach is unique in its long-term planning optimization of a network of three energy sectors simultaneously. It illustrates the effects of integrating the planning of MES on the different networks and on interactions between them. Interactions between the respective energy infrastructures are modeled and the network is optimized by considering all of the individual energy infrastructures and all of the (discrete) time periods simultaneously. The model is general, can easily be extended and is applicable over different spatial and temporal scales. It includes geographical and temporal investment decisions on energy distribution networks, - conversion, - storage and - supply, i.e., it should decide *where* and *when* to build, and its objective is to minimize long-term investment costs. Modeling investment decisions as integer variables over a discrete set of time periods and locations, and energy quantity as a continuous variable results in a mixed-integer linear program (MILP). There is a large degree of freedom: energy flow can be converted into other forms, travel to different locations or get stored and used in another time period, and investment decisions can be made in each time period and at each location. Solving this fully coupled optimization problem with state-of-the-art solvers is therefore com-

putationally challenging (Van Beuzekom et al., 2020).

Often, energy systems can be modeled as networks defined by a collection of nodes and arcs with energy flowing from node to node along paths in the network. This is also the case for the MES model that we investigate in this research. Therefore, we now introduce some fundamental concepts of network flow and network design.

2.2. Network flow and design

In this chapter, we review the fundamentals of network flow and network design, as well as previous work within these fields that is specifically interesting for our model.

Many real-life infrastructures can be represented by networks. Network design consists of, broadly defined, the planning and modification of these networks. In network flow problems, entities move from one location to another in an underlying network, and ways to do this as efficient as possible are looked into. A good understanding of both network design and network flows is crucial for designing energy systems, as many energy infrastructure design problems can be modeled as network flow problems (Gabrielli et al., 2018).

2.2.1. The minimum cost flow model

The theory of network flow is reviewed based on the textbook by Ahuja, Magnanti and Orlin (1993). We assume that the reader is familiar with the basic concepts and tools from graph theory and linear programming.

There are many applications of network flow models. A common class of network flow models consists of physical networks that arise in transportation settings and several other disciplines of science and engineering. But also for problems that might not appear to involve networks at all, network flow problems arise. For instance, many scheduling applications can be modeled as network flow problems.

The most fundamental of all network flow problems is the minimum cost flow model. In this problem, one wishes to determine a least cost shipment of a commodity through a network, while demands at nodes are satisfied from supplies at other nodes. The model has many applications, for instance in transportation- and communication networks, distribution problems, scheduling, and so on. Special versions of the minimum cost flow problem include:

- Shortest path problem
- Maximum flow problem
- Assignment problem
- Transportation problem
- Circulation problem

Ahuja et al. introduce a standard mathematical programming formulation of the minimum

cost flow problem. Let $G = (N, A)$ be a directed network, where N is a set of n nodes and A is a set of m directed edges. Each edge $(i, j) \in A$ has an associated cost c_{ij}, denoting the cost per unit flow on that edge, a capacity u_{ij} that denotes the maximum amount that can flow on the edge and a lower bound l_{ij} that denotes the minimum amount that must flow on the edge. For each node $i \in N$, $b(i)$ denotes its supply/demand. Nodes i where $b(i) > 0$ are called *supply nodes*; nodes i where $b(i) < 0$ are called *demand nodes* (where the demand is equal to $-b(i)$); and nodes where $b(i) = 0$ are *transshipment nodes*. The edge flows are the decision variables in the minimum cost flow problem. Let x_{ij} be the flow on edge $(i, j) \in A$. The minimum cost flow problem is formulated as follows:

Minimize

$$\sum_{(i,j)\in A} c_{ij} x_{ij} \tag{2.1a}$$

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b_i \qquad \forall i \in N, \tag{2.1b}$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \qquad \forall (i,j) \in A, \tag{2.1c}$$

where $\sum_{i=1}^{n} b(i) = 0$.
Constraints (2.1b) are referred to as *mass balance constraints*: the total outflow minus inflow must equal the supply/demand of every node. For a supply node, the outflow exceeds the inflow; for a demand node, the inflow exceeds the outflow; and for transshipment nodes, the outflow equals the inflow. Constraints 2.1c are referred to as *flow bound constraints*. If lower bounds are not stated for a specific problem, it is assumed that the value of the lower bounds is zero.

Generalized flow problems

In the minimum cost flow problem as formulated above, it is assumed that edges conserve flows. In some applications however, this assumption is violated. In generalized flow problems, edges are allowed to consume or generate flow. If x_{ij} units of flow enter an edge (i, j), then $\mu_{ij} x_{ij}$ units of flow arrive at node j. μ_{ij} is a positive *multiplier* (gain/loss factor) associated with the edge (Figure 2.1). It is a loss factor if $0 < \mu_{ij} \leq 1$ and a gain factor if $\mu_{ij} > 1$. In the “classical” minimum cost flow problem, μ_{ij} is equal to 1. The generalized flow problem is formulated as follows:

Minimize

$$\sum_{(i,j)\in A} c_{ij} x_{ij} \tag{2.2a}$$

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} \mu_{ji} x_{ji} = b_i \qquad \forall i \in N, \tag{2.2b}$$

$$0 \leq x_{ij} \leq u_{ij} \qquad \forall (i,j) \in A, \tag{2.2c}$$

Without loss of generality, it is assumed that the lower bound on every edge flow is equal to zero. Generalized flow problems arise in several applications, such as power transmission

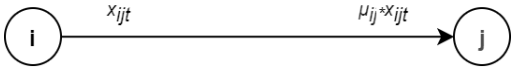


Figure 2.1: In the generalized flow problem, flow on edge (i, j) gets multiplied with a factor μ_{ij}

through electric lines, with power lost with distance traveled; or flow of water in pipes that lose water due to seepage.

Generalized flow problems are often significantly more complex than classical network flow problems when using network optimization solution methods, as some “nice” properties of regular network flow problems cannot be extended to the generalized flow case. First, the total supply does not necessarily equal the total demand in generalized flow problems. In addition, integrality properties of classical network flow problems do not extend to the generalized case if the gain/loss factors are not integer-valued.

2.2.2. Network design problems

Sometimes we are not only interested in the optimal flow in a network, but also (or particularly) in its optimal design. Network Design or Topology Design problems are network flow problems in which the set of “installed” nodes or links is optimized in a network. The objective is to minimize the total cost of the system, i.e., the sum of the design costs and the routing costs. Typically, these problems appear in long-term network planning projects, such as warehouse location, hub location, and so on. *Greenfield planning* is a planning study in which a network is built from scratch and *Brownfield planning* is a study in which a network exists and nodes/edges can be added or upgraded.

In a network design model, commodities also have to be routed on arcs between different points of origin and destination. In addition to a unit (variable) cost for routing the flow, a fixed cost for using an arc can be imposed. Because these design variables involve choices from a discrete set of values, the network design problem (NDP) can be modeled as a (mixed) integer

linear program. Magnanti et al. (1984) propose the following integer programming-based formulation for the general network design model, for a network $G = (N, A)$. Let K denote the set of commodities k. Let R_k denote the required amount of flow of commodity k to be shipped from its point of origin $O(k)$ to its point of destination $D(k)$. There are two types of variables: one modeling discrete choice decisions and one modeling continuous flow decisions. Let y_{ij} denote a binary variable, indicating whether arc (i, j) is chosen as part of the network design, and let f_{ij}^k denote the amount of flow of k on (i, j). The (mixed) integer linear program for the NDP is then stated as follows, where the set S includes all the side constraints on either the flow or the design variables:

Minimize

$$\sum_{(i,j)\in A} \sum_{k\in K} c_{ij}^k f_{ij}^k + \sum_{(i,j)\in A} F_{ij} y_{ij} \tag{2.3a}$$

subject to

$$\sum_{\{j:(i,j)\in A\}} f_{ij}^k - \sum_{\{j:(j,i)\in A\}} f_{ji}^k = \begin{cases} R_k, & \text{if } i = O(k) \\ -R_k, & \text{if } i = D(k) \\ 0, & \text{otherwise} \end{cases} \qquad \forall k \in K \tag{2.3b}$$

$$\sum_{k\in K} f_{ij}^k \leq K_{ij} y_{ij} \qquad \forall (i,j) \in A \text{ and } k \in K \tag{2.3c}$$

$$(f, y) \in S \tag{2.3d}$$

$$f_{ij}^k \geq 0, \quad y_{ij} \in \{0, 1\} \quad \forall (i,j) \in A, \forall k \in K. \tag{2.3e}$$

Interactions between design decisions and routing of flow can be considered in these types of models, as the optimal solution minimizes the combination of the per unit (variable) flow routing costs c_{ij}^k and the fixed arc design costs F_{ij}. The flow costs depend on the total arc flow volume, and the fixed charge costs depend on whether the arc is "opened", i.e., whether capacity is installed on the arcs. Constraints (2.4b) are the usual network flow conservation equations. Constraints (2.3c) state that the total flow $f_{ij} = \sum_{k\in K} f_{ij}^k$ of all of the commodities k cannot exceed the capacity of each edge if the edge is chosen as part of the network design, and that the capacity of an edge is zero if it is not chosen as part of the design.

Magnanti et al. (1984) show the intimate connection between the general network design problem and other well-known network flow problems, such as the shortest-path problem, vehicle routing problem, the traveling salesman problem, facility location models, minimum spanning trees and several other combinatorial optimization problems. They state that minor modifications to the model can make it far more complicated to solve. Assumptions on

the (un)directness of the arcs, an (in)complete demand or on the amount of sources can affect our ability to solve network design problems. Magnanti et al. show the NP-hardness of the general NDP and state that there is extensive empirical evidence suggesting that large scale (50-100) NDPs are extremely difficult to solve. When, for instance, budget constraints are taken into account, even finding a near-optimal solution in an efficient manner is most of the times very difficult.

Commonly used for modeling network design problems is the instance of the NDP with uncapacitated arc capacitites, linear routing costs, fixed opening costs and no side constraints. This problem is often referred to as the *fixed charge design problem* and is known to be an NP-hard problem (Holmberg and Yuan, 1998). In Section 2.3, solution methods for some of these network design problems are discussed.

The network loading problem

A special version of the NDP, the network loading problem (NLP), models the design of networks for which the variable flow costs are zero and facilities of fixed capacity are available to carry flow. Integer amounts of these facilities can be loaded on arcs of the network. In the NLP, the total costs for loading facilities on each of the arcs is minimized, where a given demand has to be met. In Figure 2.2, it is illustrated what the cost function for flow x looks like in an instance where there is one type of facility that can be loaded on the arcs.

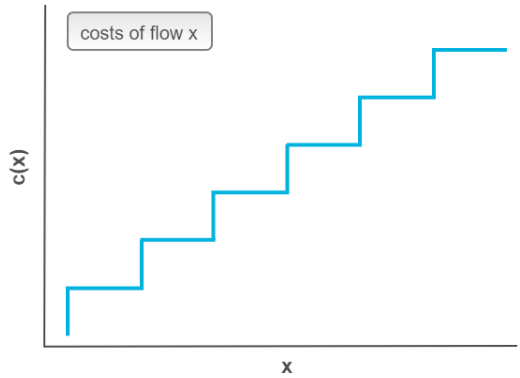


Figure 2.2: The cost function for sending flow x over a link in the network loading problem. In this instance, facilities all have the same price and capacity.

Magnanti et al. (1991) modeled the network loading problem with two facilities (the Two Facility Loading Problem or TFLP). A piecewise staircase form is assumed for the fixed costs and no costs are assumed for the routing of flow. The TFLP is modeled as follows:

Minimize

$$\sum_{(i,j)\in A} (a_{ij}x_{ij} + b_{ij}y_{ij}) \tag{2.4a}$$

subject to

$$\sum_{\{j:(i,j)\in A\}} f_{ij} - \sum_{\{j:(j,i)\in A\}} f_{ji} = \begin{cases} -R_k, & \text{if } i = O(k) \\ R_k, & \text{if } i = D(k) \\ 0, & \text{otherwise} \end{cases} \qquad \forall i \in N, \text{ for all } k \in K \tag{2.4b}$$

$$\sum_{k\in K} (f_{ij}^k + f_{ji}^k) \leq x_{ij} + C y_{ij}^k \qquad \forall (i,j) \in A \tag{2.4c}$$

$$0 \leq f_{ij}^k, x_{ij}, y_{ij} \in \mathbb{Z}^+ \quad \text{for all } (i,j) \in A, \text{ for all } k \in K \tag{2.4d}$$

In the above formulation, N and A denote the set of nodes and arcs of the network respectively, and K denotes the set of commodities. Commodity k has origin $O(k)$, destination $D(k)$ and demand d_k. There are two types of facilities considered: low capacity (LC) and high capacity (HC) facilities. The LC facility has capacity 1 and the HC facility has capacity C. Design variables x_{ij} and y_{ij} define the number of LC and HC facilities that are loaded on arc (i,j), and a_{ij} and b_{ij} represent the costs of loading a single LC or HC facility. Flow variables f_{ij}^k model the flow of commodity k on arc (i,j). The objective function minimizes the total cost for loading all of the facilities.

From a computational complexity point of view, the TFLP is difficult. By reducing the strongly NP-complete 3-partition problem to the decision version of the TFLP, Magnanti et al. show that the TFLP is strongly NP-hard. In Chapter 3, we prove in a similar way that this also holds for our problem.

Multi-period extensions

Most of the work carried out in the field of network design concerns static models, meaning that an optimal design is looked for when the requirements at a specific time instance are considered. The evolution of requirements over a longer time period is not taken into account in these models. A possible way of considering this evolution over time is to solve the static model at the consecutive time instances $t = 0, 1, \ldots, T$ of the time period $[0, T]$, i.e., finding a sequence of optimal static solutions. However, as pointed out in a paper by Minoux (1987), because of the discontinuous nature of possible investment decisions, examples are easily found where such a sequence is not only far from the optimum, but also practically infeasible. Minoux states that good investment policies can only be obtained by taking the dynamic nature of network optimization problems into account. Still, little attention has been given to multi-period extensions of network design problems, likely due to the intrinsic complexity of realistic dynamic models.

Most studies on multi-period network design consider problems in the telecommunication networks. Even though the problem settings are different, many of these models apply equally well to energy infrastructure design models, as the underlying mathematical models are often the same. Minoux (1987) was the first to study the complexity and the dynamic nature of the multi-period NDP. In this research, a dynamic NDP to investigate network expansion over a given time period $[0, T]$ is presented. The time period is divided into T intervals of unit duration and it is required that, at each time instance, the installed capacity on each link is sufficient to meet the traffic requirements of that link. Once an investment is made, it is not reconsidered later on. Link capacitites are therefore considered as non-decreasing functions of time. The resulting dynamic NDP is highly combinatorial, and finding exact solutions is extremely difficult. Efficient approximation algorithms have to be devised to approximate near-optimal solutions. Some of these algorithms for different variations of the multi-period network design problem are discussed in Section 2.3.

2.3. Solution Methods

In this section, we discuss various solution methods for network design problems. We explicitly focus on introducing methods that are relevant for solving our MES problem.

For a few special cases of the NDP, there are efficient optimization methods (Magnanti and Wong, 1984). For instance, a fixed charge design problem with zero routing costs, uncapacitated arcs and a complete demand (i.e., a demand between each pair of nodes), reduces to a minimal spanning tree problem. A fixed charge design problem with zero fixed costs, on the other hand, reduces to a shortest path problem for each commodity. It is well-established that very efficient algorithms exist for the shortest path problem and the minimal spanning tree problem. Therefore, whenever routing costs or design costs are dominant, and the network is undirected and uncapacitated, the problem becomes relatively easy to solve. The addition of various side constraints can make special purpose network flow programming methods inapplicable. However, by exploiting substructures of such a problem with complicating side constraints, one might find subproblems that *can* be solved using network flow programming algorithms. In this method, problems are *decomposed* into subproblems that can be solved as stand-alone models using efficient special purpose algorithms. In more general and complex cases of the NDP, mathematical programming methods are necessary for obtaining exact solutions (Magnanti and Wong, 1984). Since every network design model with linear functions has a linear programming model, it is possible to solve them with general purpose linear programming algorithms. When there are no integer variables and the NDP can be formulated as an LP, the simplex method can be applied. For a detailed explanation of the simplex method, we refer to (Ahuja et al., 1993). When there are integer variables present, branch and bound methods are among the most effective techniques for general and complex cases of the NDP.

2.3.1. Branch and bound methods

For many optimization problems, direct solution methods might not exist, or they might be highly inefficient. Branch and bound methods enable us to solve these “difficult” problems

by applying existing methods to “easy” subproblems. By solving the easy subproblems, one hopes to find a solution for the original problem.

General branch and bound

In general branch and bound methods, optimal solutions are found through a structural search of the space of all feasible solutions (Lawler and Wood, 1966). In the search process, which is represented by a tree of subproblems, the space of feasible solutions is iteratively partitioned into smaller and smaller subsets (nodes) and lower bounds are calculated within each subset. At some nodes, the corresponding subproblem can be split into smaller subproblems, creating child nodes in the tree. Nodes with a bound that exceeds a known feasible solution are excluded from further partitioning (pruned). The algorithm keeps track of the best solution found during the search, the incumbent, which can be updated throughout the whole search. When the complete tree is visited, the partitioning terminates, and the incumbent equals an optimal solution. The number of computations is related to the number of distinct bounding problems created, i.e., the number of nodes in the tree.

For the remainder of our explanation of branch and bound methods, we will only consider how they are applied for solving MILPs. For a complete overview of applications of branch and bound methods, within and outside of the domain of mathematical programming, we refer to Lawler and Wood (1966). Unless stated otherwise, we consider minimization problems.

In the case of (mixed) integer linear programs, the “easy” subproblems are linear programs, that we can solve using the simplex method. Branch and bound methods continuously solve linear relaxations, i.e., the version of the problem in which the integrality constraints are relaxed, and keep track of the best current feasible solution (incumbent). The root node of the tree is the linear relaxation of the original problem. If all original integer variables have an integer value in the optimal relaxed solution, an optimal integer solution to the original problem is found. If this is not the case, then an integer variable with a noninteger value is chosen and restricted to be lower than the rounded down non-integer value or higher than the rounded up non-integer value. As an example, let x be an integer variable which equals 3.3 in the LP relaxation of the original problem. Then, two subproblems (child nodes) are created in which $x \leq 3$ and $x \geq 4$. On both of these problems, the process is repeated.

For the LP relaxation in a node in the search tree, there are several possibilities.

- The LP relaxation is infeasible. If this is the case, then the corresponding node will be pruned.
- The objective value of the LP relaxation is larger than the incumbent. If the objective value of the relaxation is larger than the incumbent, then the node cannot yield a better integral solution and will therefore be pruned.
- The objective value of the LP relaxation is better than the incumbent. In this case, two new child nodes will be created.

- All the integrality restrictions of the original MILP are satisfied in the LP relaxation. A feasible solution is found and the node becomes a permanent leaf on the search tree: it is no longer necessary to branch on it.

Best bound and gap The objective value of the incumbent is a valid upper bound (denoted by Z^U) on the optimal solution of the MILP. There is also a valid lower bound (Z^L), which is obtained by taking the minimum of all the objective values of the current leaf nodes. The difference between the current upper and lower bounds is known as the gap (denoted by $\gamma(Z^L, Z^U)$). It is computed as follows

$$\gamma(Z^L, Z^U) = \begin{cases} 0, & \text{if } |Z^L| = |Z^U| = 0, \\ \dfrac{|Z^U - Z^L|}{\max\{|Z^L|, |Z^U|\}}, & \text{else} \end{cases} \tag{2.5}$$

When the gap is zero, optimality is demonstrated.

Branch and cut Cutting plane methods solve linear relaxations of MILP's and use violated constraints in the original problem to generate cuts (Winston and Goldber, 2004). These cuts are additional constraints in the next iteration and the process is repeated until an optimal integer solution is found. Branch and cut methods combine both branch and bound and cutting plane methods. They are known to have the ability to solve large MILP's in reasonable time (Matuschke et al., 2014) and lie at the heart of most state-of-the-art MIP solvers.

Since many NDPs are extremely difficult to solve, it can be helpful to strengthen the branch and bound or branch and cut methods. There are multiple ways of strengthening these methods, and we highlight a few here.

Valid inequalities Within a branch and bound or branch and cut algorithm, valid inequalities (VI) can be added to the initial formulation of the MILP. This formulation has a better initial LP relaxation and is often easy to implement. However, when considering VI, a trade-off has to be made between a tighter LP relaxation and more difficult subproblems due to the addition of constraints.

Heuristic methods When solving MILPs with branch and bound, it is extremely valuable to have good incumbents and to find these as quick as possible. There are a few reasons for this. Sometimes it might not be possible to solve a problem to provable optimality, for instance if the underlying MILP is too difficult or if the solver can only run for a restricted amount of time. Good incumbents help the solver to end up with the best possible solution at termination. It is therefore valuable to do a little extra work in some of the nodes of the search tree to see if a good integer feasible solution can be extracted.

2.3.2. Decomposition methods

Various network design problems, however difficult they might be to solve, can have an attractive substructure. Network design problems with variable flow costs, for instance, have

shortest path problems as embedded network structure. When solving NDPs, it is therefore often interesting to exploit their substructure algorithmically. In literature, many examples can be found of decomposition based approaches that are applied to network design problems. Among these, the Lagrangean relaxation method and Benders decomposition appear common methods in the field of network design, testified by a large body of literature as will be discussed next. In Chapter 6, we provide a detailed explanation of these respective methods.

Lagrangean relaxation In literature, many examples can be found of Lagrangean relaxation based approaches that are applied to network design problems. Studies on (static) fixed charge network design often propose a Lagrangean relaxation approach, combined with subgradient methods and branch and bound methods (Gendron and Crainic, 1998; Holmberg and Yuan, 1998, 2000). In all of these works, it it stated that Lagrangean relaxation used alone is insufficient to solve large-scale and difficult instances and that it works best when combined with primal heuristics. Also the class of multi-period network design problems is an attractive candidate for Lagrangean relaxation. Various studies on multi-period network design problems successfully applied variations of Lagrangean relaxation, again often in combination with other methods, to find approximate solutions to the problem (Bernard Fortz and Enrico Gorgone, 2012; Chang and Gavish, 1995a; Dutta and Lim, 1992; Fragkos et al., 2017; Kubat and Smith, 2001; Quelhas et al., 2007). Magnanti et al. (1993) apply a Lagrangean relaxation strategy to the network loading problem and show that their bound is stronger than the LP relaxation in the case of the two-facility loading problem (TFLP). However, in another paper by Magnanti et al. (1991), it is showed that for the TFLP, a linear programming formulation that includes valid inequalities always approximates the value of the MILP at least as well as the Lagrangean relaxation bound. Dutta en Lim (1992) consider a joint multi-period capacity and flow allocation problem on a network graph with internode traffic requirements for each period. As their problem is combinatorially explosive, they propose a method for finding approximate solutions based on Lagrangean relaxations. The method finds relatively good solutions to the problem for instances of 30 nodes and six time periods. Chang and Gavish (1995b) suggest in their paper tight lower bounding procedures for a multiperiod network expansion problem for a telecommunications network. For this, they develop a heuristic that combines Lagrangean relaxation with the addition of valid inequalities, and find that good solutions can be found for various network instances.

Benders decomposition A survey provided by Costa (2005) presents an overview of applications of Benders decomposition for fixed charge network design problems. Benders decomposition was usually applied to problems containing one set of integer variables, associated with the arcs in the network, and one set of continuous variables, associated with commodity flows. Costa stated that Benders decomposition is an efficient method for solving network design problems, and may outperform other techniques such as Branch-and-Bound or Lagrangean relaxation. Gabrel et al. (1999) propose an exact constraint generation approach for solving network optimization problems with general discontinuous step-increasing cost functions. The procedure that they propose is a specialization of the Benders procedure. They were the first to perform a systematic computational study providing exact optimal so-

lutions to this class of network optimization problems and they confirm that standard LP software can be practically applied for solving these. Gascon et al. (1993) combine Benders decomposition and Lagrangean relaxation for a multi-period network design problem, where Lagrangean relaxations are used to solve the master problem. They found that their combined method outperformed branch and bound methods. Fragkos et al. (2017) propose a formulation for a multi-period multicommodity network design problem, capturing the time-dependent network decisions. They develop a custom heuristic tailored to large-scale, multi-period problems. For capacitated variants of the network design problem, they develop an arc-based Lagrangean relaxation, combined with local improvement heuristics. For uncapacitated variants, several Benders decomposition variants are developed. Computational results show that both Lagrange relaxation and Benders decomposition are efficient in finding near-optimal solutions within a reasonable amount of time.

2.4. Complexity classes

In this section, we provide an introduction to computational complexity, based on Pinedo (2016) and Karp (1975). In the study of computational complexity, problems are classified on how "difficult" they are to solve. The term *problem* refers to the description of the abstract question to be solved. An *instance* is a problem with a given set of data. The *size* of an instance refers to the length of the data string that is necessary to specify the data as input for the problem. Measuring the difficulty of a problem can be done by examining the worst-case number of computational steps that a Turing machine requires to solve it optimally. This number of steps is proportional to the size n of an instance, and is therefore calculated as a function $T(n)$ of the instance size. If $T(n)$ is a polynomial, then we say that the problem can be solved optimally in *polynomial time*.

Within the framework of computational complexity, we distinguish between *decision problems* and *optimization problems*. Decision problems contain a question to which the answer is "yes" or "no", whereas an optimization problem consists of finding the best solution from a set of feasible solutions. For each optimization problem, a decision problem can be defined. An optimization problem and its corresponding decision problem are strongly related: if there exists a polynomial time algorithm for the optimization problem, then there exists a polynomial time algorithm for its decision problem and the other way around.

Many optimization problems are shown to be *equivalent*, in the sense that either all of them or none of them can be solved in polynomial time. A complexity class is a set of problems with a similar complexity. We formally introduce the following problem classes.

Definition 1 (Class $\mathcal{P}$)**.** *The complexity class $\mathcal{P}$ is the set of all decision problems for which a Turing machine algorithm exists that, for each instance, leads to the right answer in polynomial time (or equivalently, in a number of steps that is bounded by a polynomial function of the instance size).*

In the definition of the class $\mathcal{P}$, we consider the time a Turing machine requires to *solve* a decision problem. A larger class of decision problems considers the time a Turing machine re-

quires to *verify* whether a proposed solution (*certificate*) is correct or not.

Definition 2 (Class $\mathcal{NP}$)**.** *The complexity class $\mathcal{NP}$ is the set of all the decision problems for which the correct answer can be verified by a Turing machine algorithm within polynomial time when given a certificate.*

It is obvious that $\mathcal{P} \subset \mathcal{NP}$: if a solution can be found in polynomial time, then the solution can be verified in polynomial time. One of the most important unsolved problems in mathematical logic and combinatorial optimization is whether or not $\mathcal{P} = \mathcal{NP}$. If this would be the case, then there would exist polynomial time algorithms for a very large class of problems for which, up until now, no polynomial time algorithms have been found.

Combinatorial problems are often either special cases of other problems, more general than other problems, or equivalent to other problems. Therefore, an algorithm that works well for one combinatorial problem often works well for many other problems after minor modifications. To show the equivalence of problems, we make use of the concept of problem reduction. We say that decision problem P *(polynomally) reduces* to P' if there exists a (polynomial time) function that translates any instance of P into an instance of P'.

We can now introduce the class of NP-hard problems.

Definition 3 (NP-hardness)**.** *A decision- or optimization problem P is called NP-hard if every problem in $\mathcal{NP}$ polynomially reduces to P.*

Some problems in the class of NP-hard problems are more difficult than others. The set of *strongly NP-hard* problems is the set of problems that remain NP-hard, even if all the numbers in the input are bounded by some polynomial in the length of the input. Other NP-hard problems can be solved with polynomial time algorithms with such an input: the problems that are *NP-hard in the ordinary sense* or simply *NP-hard*.

Lastly, we introduce the class of NP-complete problems.

Definition 4 (NP-completeness)**.** *A decision problem P is called (strongly) NP-complete if 1) P is (strongly) NP-hard and 2) $P \in \mathcal{NP}$.*

If any NP-complete problem has a polynomial time algorithm, then all problems in $\mathcal{NP}$ do.

3

The Multi-Energy System Design Problem

In this chapter, we formally introduce the model that this research is based upon. First, we introduce the model and its elements. Second, we introduce a network flow formulation for this model and evaluate it from a network flow perspective. Third, we investigate its computational complexity.

3.1. Model description

Van Beuzekom et al. (2017) introduce a mixed integer linear program for optimizing the design of a multi-energy system which couples different energy carriers. All of these energy carriers are fully coupled at each location and in each time period. The model is intended for use at a city-scale, but could be scaled up- and downwards for different geographical scales, such as buildings or larger regions. The design of the system is defined as the infrastructure of electricity, gas and heat networks, including three types of conversion and three types of storage assets and energy suppliers. The focus lies on long-term planning, using annual time steps, although it could also be adjusted to other temporal scales. Both temporal and geographical scale adjustments would require some adjustments in the model assumptions. In this section, the setup and the different elements of the model that we examine in this research are introduced. This is a variation on the most general version of the model as introduced by Van Beuzekom et al. (2017) and focuses specifically on medium-sized cities coupling electricity, gas and heat.

Energy carriers and networks

Three energy carriers and corresponding networks are considered in the model: electricity, (natural) gas and district heating. The city is modeled as a set of different locations, each containing a demand profile. All of the locations have a demand for electricity, gas and heat in each of the time periods considered. These energy carriers all behave in a different way, but also share some similarities: as an example, transportation losses occur for each of these carriers. These losses are taken into account in the model.

Network connections transport energy between different nodes. These can either represent electricity lines, gas pipelines or heat pipelines. All networks are modeled at medium voltage or equivalent levels and network losses are linearized. The district heating network generally consists of a combination of high and lower temperature heat, but is modeled here as a single network. For each of the network connections, it is assumed that energy can flow in both directions.

Assets

Investment decisions can be made regarding network connections, and supply-, conversion- and storage technologies.

Supply The energy supply mix is based on future scenarios and climate goals for a specific city. In this case, only gas and electricity are fed into the system from suppliers; electricity through RES (wind- and photovoltaic or solar (PV) energy) and gas supply through pre-determined gas access points. The location and capacity of gas supply is known beforehand for each year and not variable (in variations on the model, the gas supply can also be declared as a decision variable). RES, on the other hand, are declared as decision variables and can be built on the different nodes.

Conversion Energy conversion units are essential for an integrated energy system. A conversion is defined as a transformation between the energy carriers within the energy network. Several technologies exist which convert one energy carrier into another. The ones that are considered in this research are a combined heat- and power plant (CHP), heat pump (HP) and power-to-gas unit (P2G). CHP technologies allow the generation of electricity and heat at the same time from a variety of fuels. In this model, CHP transforms natural gas into both heat and electricity. Heat pumps can be used as suppliers for district heating, integrating the heating, cooling and electricity networks. For simplicity, this is modeled here as the conversion of electricity into heat. Power-to-gas is a technology that converts electricity into gas, using electrolysis. In the model, a P2G unit allows for the conversion of electricity into gas. The conversion rate for each of these technologies specifies how much of a certain energy carrier you can “create” from another.

Storage Storage units are used for storing excess energy at the end of the year and supplying this to the system in the next year. For each of the energy carriers, storage possibilities are considered. The storage possibilities differ significantly in costs and efficiency: due to the large size of the time periods, large standing losses can occur. Storage technologies for electricity, for instance, are most effective at a short term, whereas heat storage can be used most effectively between seasons. Two efficiency factors are taken into account for the storage technologies: an efficiency factor for feeding energy into/taking energy from the storage units and standing losses, which is an annual percentage.

Combining all of these assets in the same system allows for energy flow to travel between locations and between time periods, respectively, or get converted into another form.

An overview of the model elements can be found in Table 3.1. Figure 3.1 provides an overview of a network for a 7-node case.

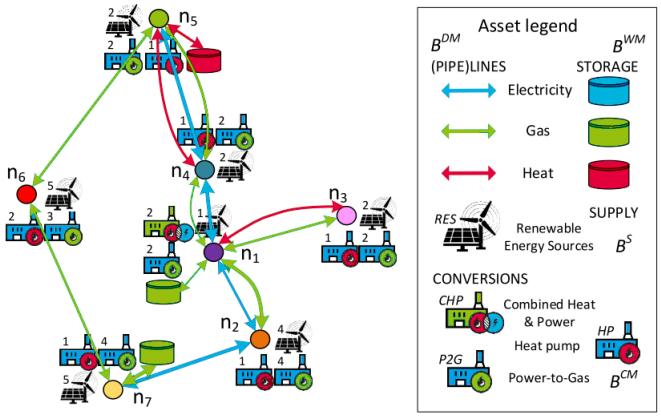


Figure 3.1: An example of results for a 7-node network (van Beuzekom et al., 2017)

Requirements

The model is constrained by sustainability objectives (modeled on the supply side), the (spatial) demand for the different energy carriers in each time period and the law of conservation of energy.

Objective

The goal of the model is to find the optimal mix and location of supply- , storage- , and conversion units and network connections such that investment costs are minimized and that all of the requirements of the system are met. The investment decisions can be made at each time step and at each location. Only the annual investment costs are considered in this model: operational costs (such as maintenance, operating time, etc) are (indirectly) included in the model as energy losses. The annual investment costs are based on a pre-determined discount rate. The model can either take the existing infrastructure into account, i.e., provide a starting solution for the first time step (Brownfield), or build a network from scratch (Greenfield).

Even though the behaviour of the individual energy systems is non-linear, the model is set up as a mixed integer linear program. Because of the large size of the time steps and the and given the much larger uncertainty in the future scenarios, the error from the linearization is relatively minor (Van Beuzekom et al., 2020).

Scope and limitations

Mixed-integer programming models are attractive in cases where a large variety of decisions are considered simultaneously (Magnanti and Wong, 1984). However, they are not well-suited for dealing with underlying uncertainties and risks. Investment strategies developed by this model should therefore ideally be tested by a simulation analysis. Especially the combination

Energy carrier	Element	Function	Data available
Gas	Gas supply	Supply gas	Capacity per location *(PJ/yr)*
	Gas pipeline	Transport (and possibly store) gas	Costs *(euro/m)* Capacity *(PJ/yr)* Loss
	Combined Heat and Power	Transform gas into heat and electricity	Costs *(euro/unit)* Capacity *(PJ/yr/unit)* Conversion rate
	Gas storage unit	Store gas	Costs *(euro/unit)* Capacity *(PJ/yr/unit)* Loss and efficiency factor
Electricity	Renewable Energy Supply (RES): PV and wind	Produce electricity	Costs *(euro/unit)* Capacity *(PJ/yr/unit)*
	Electricity line	Transport electricity	Costs *(euro/m)* Capacity *(PJ/yr)* Loss factor
	Power-to-gas	Convert electricity into gas	Costs *(euro/unit)* Capacity *(PJ/yr/unit)* Conversion rate
	Heat pump	Consume electricity for producing heat	Costs *(euro/unit)* Capacity *(PJ/yr/unit)* Conversion rate
	Electricity storage unit	Store electricity	Costs *(euro/unit)* Capacity *(PJ/yr/unit)* Loss and efficiency factor
Heat	Heat pipeline	Transport heat	Costs *(euro/m)* Capacity Loss factor
	Heat storage unit	Store heat	Costs *(euro/unit)* Capacity *(PJ/yr/unit)* Loss and efficiency factor

Table 3.1: An overview of the elements and corresponding data used in the multi energy infrastructure in the model proposed by Van Beuzekom. For all the costs, a yearly discount rate of 4% is assumed.

of the large time scale that this model considers and the short-term fluctuations in real-life supply from renewable sources illustrates the need for some “post-processing” of the optimal network. This is however beyond the scope of this research. In addition, it is assumed that integer amounts of assets can be built. To be able to solve real-case instances, coarsening of the original network should often be applied and all of the model’s parameters should be scaled accordingly. However, integrality of the investment decisions remains a constraint for each problem size. In this sense, the optimal solution for a coarser and a finer representation of the same network might differ. Finer networks are, with a fear of stating the obvious, preferred, but might be more difficult to solve. Solving the problem on a coarsened network, on the other hand, could result in large assets that are operating (far) below their optimal capacity, something that is not desired in practical cases. A possibility to deal with this is to change the decision variables into binary variables, i.e., a single fixed cost, while incurring a variable

cost for the energy flow. This is also beyond the scope of this research, as we only consider integer investment decisions.

3.2. Model formulation

In this section, we introduce a network flow formulation for the model that was introduced in Section 3.1 and evaluate it from a network perspective.

For the network flow formulation of this *multi-energy system design problem* (MESDP) , we first introduce the following sets. Given is a set of locations $l \in L$ as shown in Figure 3.2 , each with a yearly demand for each of the different energy carriers. Let $t \in T$ denote the set of different (discrete) time periods and let $k \in K$ denote the set of energy carriers, consisting of electricity, gas and heat. On each location, we can build three types of conversion- and storage units and two types of supply units. Between these locations, we can build three different network connections. To model this as a network graph, we split each location in three different nodes, by introducing a node for each of the energy carriers.

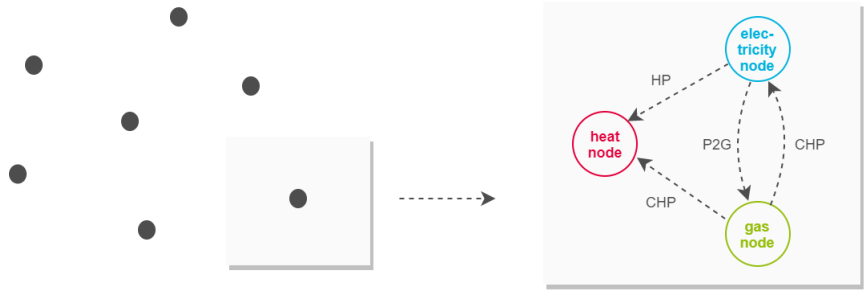


Figure 3.2: Transforming the graph of locations $l \in L$ into nodes in network graph $G(V, E)$. An instance with n locations results in a network graph with $3n$ nodes.

By introducing a node for each energy carrier (*commodity*), we can formulate the multi-commodity MESDP as a single-commodity network flow problem, i.e. a more aggregate formulation. The motivation for this is that if the number of commodities in a problem is small, an aggregated formulation has proven to be most efficient for various solution algorithms (Fragkos et al., 2017).

Let $i \in V$ denote the set of nodes, where $|V| = 3|L|$. V has the following subsets: V^e, V^g and V^h, representing the electricity- gas- and heat nodes, respectively. Each node $i \in V$ has a demand b_i for its corresponding energy carrier. To model the different conversion units, we introduce arcs between these nodes, as illustrated in Figure 3.2. Let E^C denote the arcs that represent these conversion units. To model the network connections, i.e. (pipe)lines between the different locations, we introduce E^N, the set of bidirectional arcs or *edges*. This set consists of all the possible connections within V^e, V^g and V^h and represents the possible electricity lines, gas pipelines and heat pipelines. We finally introduce network $G = (V, E)$, where $E^C \cup E^N = E$.

Mass balance constraint The first constraint for the flow in network $G = (V, E)$ is the mass balance constraint, which states that no energy is lost and that the demand of each node has to be fulfilled at each time period. Energy flow can enter node i from other nodes, i.e., via con-

verters or network connections; or from assets built on node i: either from electricity supply units (only at electricity nodes), gas supply (only at gas nodes) or storage units, i.e., excess flow from the previous time period. In each time period, flow can leave node i and travel to other nodes or get stored at node i in a storage unit.

We introduce the following variables. Let x_{ijt} denote the energy flow between node i and j at time t [1]. Let S_{it} denote the amount of flow that gets fed into the system from (gas or electricity) supply at node i at time t. Let W_{it}^{IN} denote the amount of flow that gets stored (in a "well") at node i at time t. Let W_{it}^{OUT} denote the energy flow fed into the system from the storage units at node i at time t, i.e., energy that has been stored in time periods before t. We introduce the following parameters. Let b_{it} denote the demand at node $i \in V$ at time t, where each node has a demand for its corresponding carrier (heat nodes have a heat demand, etc). Lastly, let μ_{ij} denote the arc multiplier of arc/edge (i,j): either the rate of converting the energy carrier of i into the energy carrier of j (on converter arcs), or the energy losses over the distance between i and j (on network edges).

We then find the following mass balance constraint in network $G(V,E)$:

$$\sum_{\{j:(j,i)\in E\}} \mu_{ji} \cdot x_{jit} - \sum_{\{j:(i,j)\in E\}} x_{ijt} + S_{it} - W_{it}^{IN} + W_{it}^{OUT} = b_{it}, \quad \forall i \in V, \forall t \in T \tag{3.1a}$$

Supply capacity constraints Let z_{it}^{s} denote an integer investment decision that represents the number of electricity supply units of type s (either PV or wind), built on node i at time t. Let parameter u_t^s denote the capacity of electric supply unit s at time t and let parameter g_{it} be the gas supply at node i at time t. We find the following constraint for supply S_{it}.

$$S_{it} \leq \sum_{s} \sum_{q=1}^{t} (u_t^s \cdot z_{iq}^s + g_{it}) \qquad \forall i \in V,\ \forall t \in T \tag{3.1b}$$

It is important to note that $g_{it} = 0$ for all the non-gas nodes. As g_{it} represents a parameter, we do not add this specifically in the model. In addition, (electricity) supply units can only be built on electricity nodes. Therefore, the following constraint is added:

$$z_{it}^{s} = 0, \quad \text{if } i \notin V^e \quad \forall i \in V, \forall t \in T, \forall s \in S \tag{3.1c}$$

[1] Flow variable x_{ijt} can either represent electricity-, gas- or heat flow, but because of the way that we defined network graph $G(V,E)$, we do not distinguish between different energy carriers (commodities) and can consider the problem as a single-commodity one.

Flow capacity constraints In network $G(V, E)$, the capacity of the arcs and edges at time t is dependent on the amount of assets that are built on it at time t and in each time period before t. Let z_{ijt} denote an integer investment decision that represents the number of assets (either (pipe)lines or conversion units) build on arc/edge (i, j) at time t. Each arc or edge has exactly one type of asset that can be built on it. Let u_{ij} denote the capacity of a single asset on arc/edge (i, j). We find the following constraints for the maximum flow on all of the arcs and edges:

$$x_{ijt} \leq u_{ij} \cdot \sum_{q=1}^{t} z_{ijq} \quad \forall (i,j) \in E,\ \forall t \in T \tag{3.1d}$$

Modeling the MESDP as a directed graph, we require additional constraints for the bidirectional network edges between different locations. If a network connection is built between two nodes, then flow can can be sent in two different directions. To assure that network edges are bidirectional, we add an additional constraint for the (network) edges E^N:

$$z_{ijt} = z_{jit} \qquad \forall (i,j) \in E^N,\ \forall t \in T \tag{3.1e}$$

In a feasible flow solution, flow should only travel in one direction on the bidirectional edges in each time period. To assure that this is the case, we introduce the following "forcing" constraints, where γ_{ijt} denotes a binary variable and $\mathcal{M}$ is a parameter with a sufficiently large value:

$$x_{ijt} \leq \mathcal{M}\gamma_{ijt}, \qquad \forall (i,j) \in E^N,\ \forall t \in T \tag{3.1f}$$

$$x_{ijt} \leq \mathcal{M}(1 - \gamma_{jit}), \quad \forall (i,j) \in E^N,\ \forall t \in T \tag{3.1g}$$

CHP constraints The CHP converts gas into both electricity and heat. In other words, with a single CHP investment, two different outflows are created from the gas inflow. In the network graph, this means that by building a single CHP you build assets on two arcs: from gas to heat and from gas to electricity. To assure this, we introduce the following constraint:

$$z_{ijt} = z_{iht}, \quad \forall (i,j), (i,h) \in E^C : i \in V^g, \forall t \in T \tag{3.1h}$$

An additional constraint from the "double" conversion of gas is that the amount of gas that is transformed into heat equals the amount that is transformed into electricity. We model this by forcing the flow from the gas nodes to the heat- and electricity nodes to be equal, as illustrated in Figure 3.4. We add the following constraint to the CHP edges.

$$x_{ijt} = x_{iht}, \quad \forall (i,j), (i,h) \in E^C : i \in V^g, \forall t \in T \tag{3.1i}$$

Storage constraints Let z_{it}^W denote an integer investment decision that represents the number of storage units, built at node i at time t. Let parameter u^k denote the capacity of a storage unit for energy carrier k. When energy gets stored in a certain time period, two types of losses occur. Efficiency losses ξ_k occur when energy gets stored or when energy is taken from storage units. Standing losses η_k occur for each year that the energy remains in the storage unit. The amount of flow that gets stored at node i at time t, W_{it}^{IN}, cannot exceed the available storage space, which equals the capacity of the storage unit minus the total of what is stored in it up to time t. To ensure this, we introduce the following constraint.

(3.1j)

$$W_{it}^{IN} \leq u^k \cdot \sum_{q=1}^{t} z_{iq}^W - \sum_{q=1}^{t-1} \left(\xi_k \cdot W_{iq}^{IN} \cdot \eta_k^{t-q} - W_{iq}^{OUT} \right), \quad \forall i \in \bigcup_{k \in K} V^k, \ \forall t \in T \tag{3.1k}$$

Additionally, the amount of flow that is supplied from storage units at node i at time t, W_{it}^{OUT}, cannot exceed the amount that is stored in it at that time t. Therefore, we introduce the following constraint:

$$W_{it}^{OUT} \leq \xi_k \cdot \sum_{q=1}^{t-1} \left(\xi_k \cdot W_{iq}^{IN} \cdot \eta_k^{t-q} - W_{iq}^{OUT} \right), \quad \forall i \in \bigcup_{k \in K} V^k, \ \forall t \in T \tag{3.1l}$$

Nonnegativity constraints Lastly, we define all the variables to be nonnegative, the investment decision variables to be integer and the flow direction variable to be binary:

$$z_{ijt}, \ z_{it}^s, \ z_{it}^W \in \mathbb{Z}^+, \quad x_{ijt}, \ S_{it}, \ W_{it}^{IN}, \ W_{it}^{OUT} \in \mathbb{R}^+, \quad \gamma_{ijt} \in \{0, 1\} \tag{3.1m}$$

The objective function that is constructed minimizes the total investment costs. This is equal to all the investment decisions made over time multiplied by the costs of making these decisions. Let c_{ijt}, denote the costs of building a single asset on edge/arc (i, j), i.e., a network connection or a conversion unit. Let c_t^s and c_t^W denote the costs of building a single asset on one of the nodes, i.e., a supply unit s or a storage unit. We find the following objective function:

$$\text{Minimize} \sum_{t \in T} \Big(\sum_{(i,j) \in E} z_{ijt} \cdot c_{ijt} + \sum_{k \in K} \sum_{i \in \bigcup_{k \in K} V^k} \Big(\sum_{s} z_{it}^s \cdot c_t^s + z_{it}^W \cdot c_t^W \Big) \Big) \tag{3.1}$$

We arrive at the following flow-based Mixed Integer Linear Programming Problem for the MES design problem (MESDP):

Minimize

$$\sum_{t\in T}\Big(\sum_{(i,j)\in E} z_{ijt}\cdot c_{ijt}+\sum_{k\in K}\sum_{i\in\bigcup_{k\in K}V^k}\Big(\sum_{s} z^s_{it}\cdot c^s_t+z^W_{it}\cdot c^W_t\Big)\Big) \tag{3.1}$$

subject to

$$\sum_{\{j:(j,i)\in E\}}\mu_{ji}\cdot x_{jit}-\sum_{\{j:(i,j)\in E\}}x_{ijt}+S_{it}-W^{IN}_{it}+W^{OUT}_{it}=b_{it},\quad \forall i\in V,\forall t\in T \tag{3.1a}$$

$$S_{it}\le\sum_{s}\sum_{q=1}^{t}(u^s_t\cdot z^s_{iq}+g_{it})\qquad \forall i\in V,\ \forall t\in T \tag{3.1b}$$

$$z^s_{it}=0,\quad \text{if } i\notin V^e\quad \forall i\in V,\forall t\in T,\forall s\in S \tag{3.1c}$$

$$x_{ijt}\le u_{ij}\cdot\sum_{q=1}^{t}z_{ijq},\quad \forall(i,j)\in E,\ \forall t\in T \tag{3.1d}$$

$$z_{ijt}=z_{jit},\qquad \forall(i,j)\in E^N,\ \forall t\in T \tag{3.1e}$$

$$x_{ijt}\le\mathcal{M}\gamma_{ijt},\qquad \forall(i,j)\in E^N,\ \forall t\in T \tag{3.1f}$$

$$x_{ijt}\le\mathcal{M}(1-\gamma_{jit}),\qquad \forall(i,j)\in E^N,\ \forall t\in T \tag{3.1g}$$

$$z_{ijt}=z_{iht},\quad \forall(i,j),(i,h)\in E^C:i\in V^g,\forall t\in T \tag{3.1h}$$

$$x_{ijt}=x_{iht},\quad \forall(i,j),(i,h)\in E^C:i\in V^g,\forall t\in T \tag{3.1i}$$

$$W^{IN}_{it}\le u^k\cdot\sum_{q=1}^{t}z^W_{iq}-\sum_{q=1}^{t-1}\Big(\xi_k\cdot W^{IN}_{iq}\cdot\eta_k^{t-q}-W^{OUT}_{iq}\Big),\quad \forall i\in\bigcup_{k\in K}V^k,\ \forall t\in T \tag{3.1k}$$

$$W^{OUT}_{it}\le\xi_k\cdot\sum_{q=1}^{t-1}\Big(\xi_k\cdot W^{IN}_{iq}\cdot\eta_k^{t-q}-W^{OUT}_{iq}\Big),\quad \forall i\in\bigcup_{k\in K}V^k,\forall t\in T \tag{3.1l}$$

$$z_{ijt},\ z^s_{it},\ z^k_{it}\in\mathbb{Z}^+,\quad x_{ijt},\ S_{it},\ W^{IN}_{it},\ W^{OUT}_{it}\in\mathbb{R}^+,\quad \gamma_{ijt}\in\{0,1\} \tag{3.1m}$$

In Figure 3.3 it is illustrated what the network graph looks like for one location.

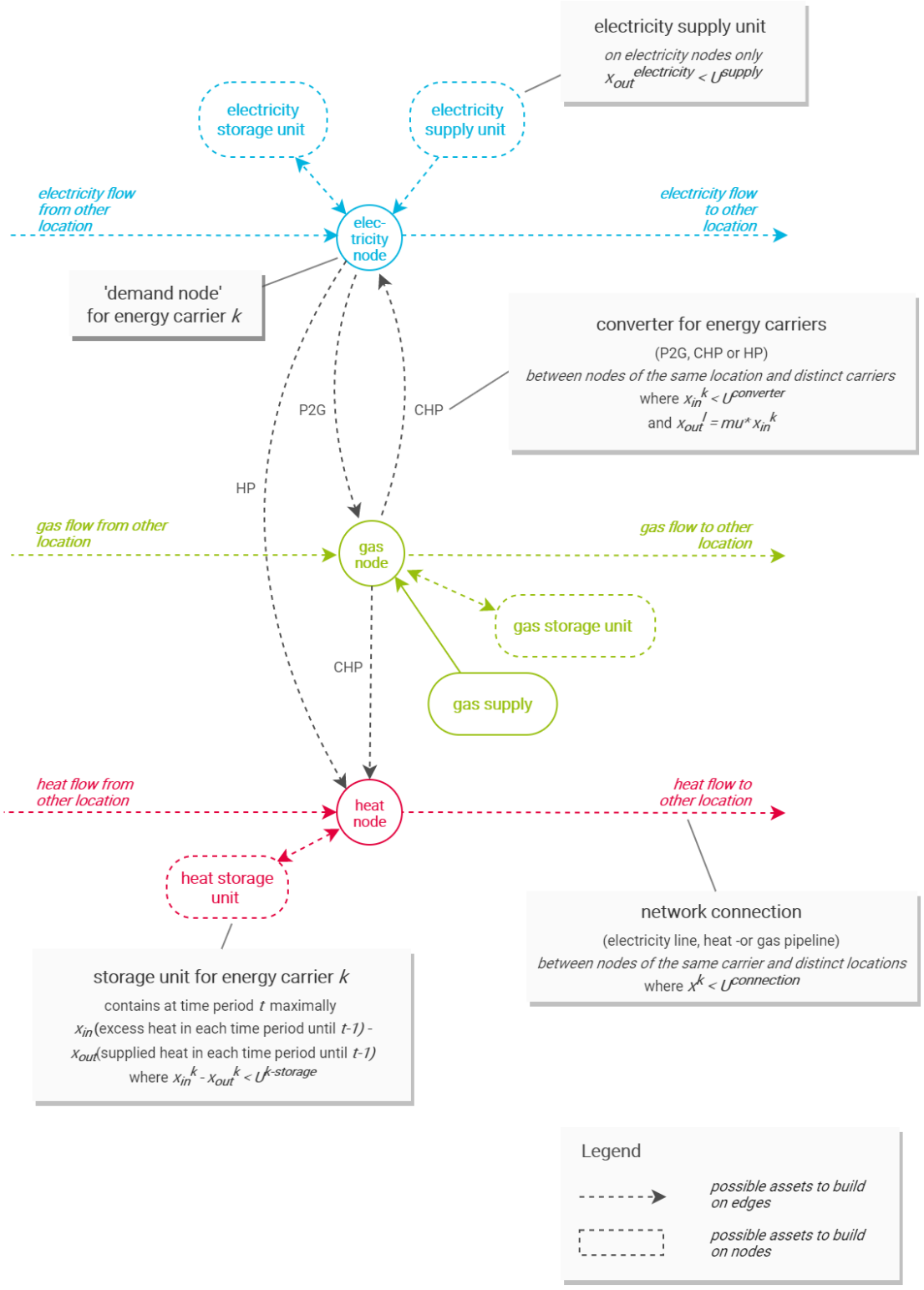


Figure 3.3: The network graph of the MESDP for one geographical location

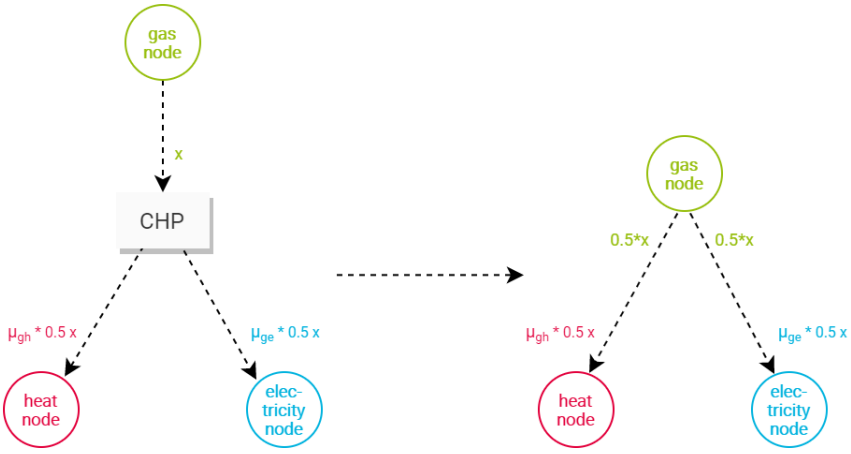


Figure 3.4: CHP units are modelled as two separate arcs from gas nodes to heat- and electricity nodes. As gas flow gets converted in both electricity and heat, we force the flow from the gas nodes to be split equally, hence the 0.5 factor in the figure. The amount of flow that arrives at the heat- and electricity nodes depends on the respective conversion rates μ.

A few additions to the introduced notation are the following:

- We have modelled the MEDSP as a single-commodity problem. This choice is motivated because there are no links possible on which multiple commodities can travel. For instance, heat and gas cannot travel on electricity lines, gas cannot be fed into a heat pump, etc. Instead of sending flows x_k for carrier $k \in K$ that can be transformed into flow of a different carrier $l \in K$, we therefore send (general) energy flow x and use the arcs multipliers μ_{ij} and the (conversion) arcs between the different node sets V^k to model the real-life transformations of the energy flow.
- To examine the influence of "bad weather years" on the network design, the time factor is taken into account for the capacity of the electricity supply units u_t^S. By varying this parameter, one can easily see the influence of the electricity supply on the final design.
- In this model, capacities are a step function of the costs: It is only possible to build assets with capacity $z_{ijt} \cdot u_{ij}$, with costs $z_{ijt} \cdot c_{ij}$ where z_{ijt} is integer valued. An example: a "single" CHP costs 8,19 million euros and it can convert 0.47 PJ ("units") of gas. Should a slightly larger CHP at a certain location be required, then the model is forced to "build" a CHP that is twice as large (and thus twice as expensive). This holds for all supply-, conversion- and storage units and all network connections in the model.

The above defined model can easily be varied to model a slightly different scenario. A few examples are:

- The storage constraints can be adjusted, for instance to force the model to store a minimum amount of energy in every year. This can help to deal with unexpected fluctuations in the supply.
- The gas supply g_{it} depends on node i: only a few locations have gas access points, each

with a pre-determined (and thus non-variable) gas supply maximum. The value of this parameter can be varied, for instance based on policy decisions.

- In variations of the model, the cost function can be defined in a different way. For instance, economies of scale could be taken into account.
- Multiple spatial or financial constraints can be added to the model. For practical cases, it might be necessary to take a total or yearly budget into account. It is probably also not possible to build all assets at each location (or in each time period); additional constraints on the investment decision variables will therefore result in more realistic problem instances.
- The model has a Greenfield planning approach when it is assumed that there is no current network and everything has to be built from scratch. Adding a "starting" solution, i.e., adding Brownfield data, makes the final design planning more realistic and could also improve the computation time, since the addition of the starting solution might decrease the solution space. In the model, existing infrastructure can be implemented by adding a z_0 term to the capacity constraints modeling the number of current assets.
- The implemented infrastructure data can also include details on the age of the individual aspects of the current network. This is relevant because of the large timescale considered in the model: it is probably not the case that each part of the current network can be used throughout the full time period considered.

Most of these additional elements are beyond the scope of this research. We conduct experiments with a Brownfield network, with various problem sizes and parameter variations, and with a few extra restrictions on the investment variables.

3.3. Problem characteristics

In this section, we discuss the characteristics of the MESDP and analyze which of these characteristics can be a (strongly) determining factor for its difficulty. Kallrath (2008) defines *difficult optimization problems* as *"problems that cannot be solved to optimality or to any guaranteed bound by any standard solver within a reasonable time limit."*

Large instances

First of all, in practical implementations of the model we are dealing with very large instances. A case study for 3 energy carriers with n locations and T time periods results in T network graphs with $3n$ nodes, $4n$ arcs and $3 \cdot \binom{n}{2}$ edges. In these networks $G(V, E)$, it is possible to build assets in various manners: storage and supply units can be built on the set of nodes, converters and pipe(lines) can be built on the set of arcs and edges respectively, and additionally, all of these decisions can be made in the different discrete time periods. On this network we are therefore solving a problem with $T \cdot (13n + 6 \cdot \binom{n}{2})$ integer variables, $T \cdot (6 \cdot \binom{n}{2})$ binary variables and $T \cdot (13n + 6 \cdot \binom{n}{2})$ continuous variables. As an example: a case study with 110 locations and 17 time periods results in a MILP with a total of 1,247,290 integer variables and 635,800 continuous variables: a highly combinatorial problem. [2]

A multi-period network loading problem

Second of all, because only integer investment decisions z^s_{it}, z^W_{it} and z_{ijt} can be made, the MESDP turns out to be an instance of the strongly NP-hard network loading problem. In addition to that, the MESDP considers multiple time periods in the design problem, which further complicates solving it. For instance, if we would consider only one time-period in the MESDP, we could model a cost function for flow x_{ijt} as a discontinuous step function, instead of optimizing on the integer investment decisions. A few studies on network optimization problems with these kinds of cost functions have been performed, as discussed in the previous chapters. However, the multi-period element makes it difficult to follow a similar approach as these previous studies, as the possibility to increase existing capacity makes an accurate cost function for flow x_{ijt} in the MESDP more complex. If a number of assets are built on edge (i, j) at time t, then those assets can also be used in the time steps following t. In a way it would be "cheaper" to send flow in those later time periods, as less additional capacity has to be built. Therefore the cost function for flow depends on previous capacity expansion decisions. Studies on multi-period network design on the other hand, for which we again refer to the previous chapter, rarely consider step-cost functions within the time periods. Since multi-period network design problems are also a very difficult class in itself, we are dealing with a combined problem that is extremely difficult to solve.

[2] In realistic scenarios, this number will probably be lower: not all connections can be built and it is also not possible to build storage-, supply- or conversion assets on each location. However, for now we assume that there are no restrictions on building assets or connections.

Storage

A difference with most of the studies on multi-period network design problems, is the flow's ability to travel to another time period in the MESDP. This is due to the use of storage assets, that can store excess flow in one time period and discharge this flow (one of) the next one(s). This significantly increases the solution space, as more trade-offs have to be made in certain time periods, for instance between building supply or storage, so excess flow from previous periods can be used.

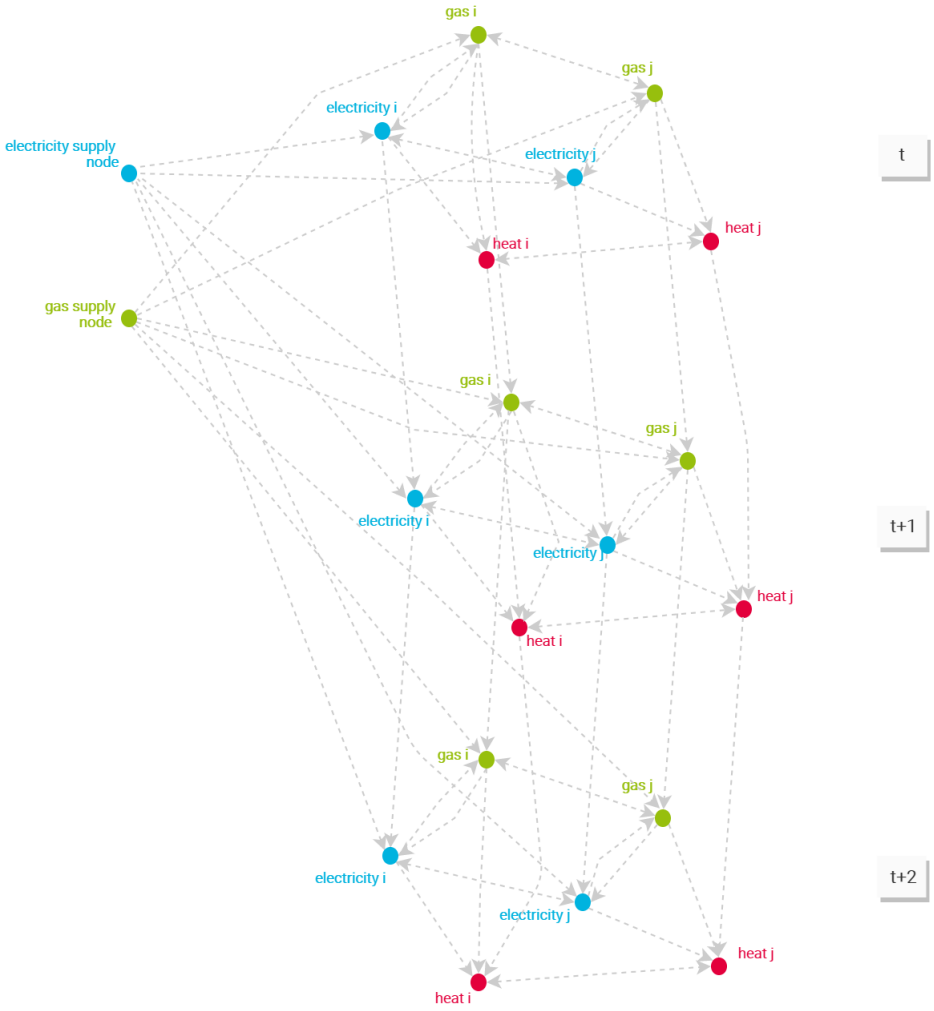


Figure 3.5: Network $G^T(V^T, E^T)$ represents an expanded network for $G(V, E)$, where only investment decisions on arcs or edges can be made, for two locations i and j.

The flow's ability to travel between time periods in the MESDP can be modeled by expanding network $G(V, E)$ into the time-expanded network $G^T(V^T, E^T)$ (Figure 3.5). By copying network $G(V, E)$ for each time period and adding edges for supply and storage investment decisions, design decisions can only be made on network edges/arcs. Electricity- and gas supply "super nodes" are introduced as sources node for network $G^T(V^T, E^T)$. Finding a feasible flow in time-expanded network $G^T(V^T, E^T)$ corresponds to finding a feasible solution for the MESDP.

In Chapter 5, we make use of this fact for deriving valid inequalities for the MESDP.

Generalized flow

The MESDP is a network problem with generalized flow, as the conversion rates and distance loss factors are multipliers on the arcs and edges respectively in the network in each time period t, and the loss factors of the storage units are multipliers of arcs in the time-expanded graph of the network. Regular network flow models have the so-called integrality property, meaning that their optimal solutions are integer if all the underlying data is integer (Ahuja et al., 1993). This property does not hold for generalized network flow models, due to the non-unimodularity of their constraint matrices. Therefore, to find integer solutions for generalized flow problems, methods of (mixed) integer programming should be applied, even if we were to provide the model with integer data only.

The MESDP consists of various complicating elements, which is why many well-established network optimization techniques are not applicable to the model. Predicting the effect of these elements, together or individually, on solution times when applying to mathematical programming methods is very difficult. Empirical evidence is therefore required to assess the quality of solvers and create a roadmap for a solution approach. An additional important factor for assessing the difficulty of the model is the its computational complexity, which we investigate in the next section.

3.4. Computational complexity of the MESDP

We show in this section that the MESDP is strongly NP-hard, by providing a reduction from an instance of a strongly NP-complete problem to an instance of the MESDP. For this, we introduce the 3-partition problem (3-PART).

Definition 5 (3-partition)**.** *Given $3n+1$ integers $a_1, a_2, \dots a_{3n}$ and B satisfying $\sum_{i=1}^{3n} a_i = nB$, does there exist a partition of $a_1, a_2, \dots a_{3n}$, consisting of n sets $S_1, S_2, \dots S_n$, each of cardinality 3, that satisfies the property that $\sum_{j \in S_i} a_j = B$ for all S_i, where $1 \leq i \leq n$?*

Lemma 1. *The 3-partition problem is NP-complete in the strong sense.*

The proof of Lemma 1 can be found in the textbook by Garey and Johnson (1979).

To show that the MESDP is strongly NP-hard, we reduce an instance of 3-PART to an instance of the MESDP with a single time period and two energy carriers (gas and electricity). This reduction has similarities with the reduction of 3-PART to the TFLP, introduced in the previous section (Magnanti et al., 1991). We can state the following.

Theorem 1. *The MESDP is strongly NP-hard.*

Proof. Given an instance $\mathcal{I}_{3-PART}$ of 3-PART, we construct an instance $\mathcal{I}_{MESDP}$ of the above mentioned special case of the MESDP as follows. For every integer $a_i, i \in \{1, 2, \dots, 3n\}$, introduce a location $i \in L$, with electricity demand $b_i^e = a_i + M$ for sufficiently large M, and gas

demand $b_i^g = 0$. Figure 3.6 shows the network graph $G(V, E)$, as defined in the previous chapter, for this particular instance of the MESDP. To graph $G(V, E)$, we introduce a root node r: building assets on the arcs between r and the other nodes corresponds to building electricity supply units. Gas supply g_i is equal to zero.

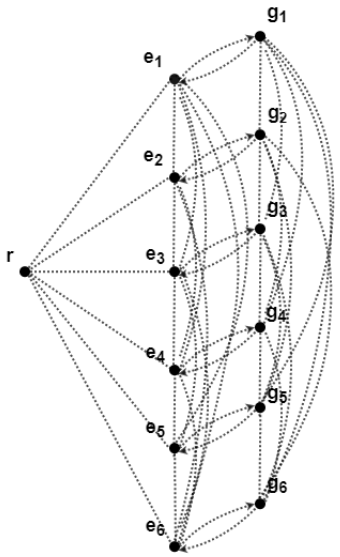


Figure 3.6: Network graph $G(V, E)$ for instance $\mathcal{I}_{MESDP}$ of the MESDP, where n is equal to 2. Connections exist between all of the electricity nodes, all of the gas nodes, and between the electricity- and gas nodes at the same location.

Flow travelling between electricity nodes of location i and j in graph G corresponds to either 1) flow travelling over an electricity line between i and j, or 2) flow travelling over a conversion arc to a gas node at location i, then over a gas pipeline to the gas node at location j and lastly over another conversion arc to the electricity node at location j. In Figure 3.7a, this is illustrated for the electricity nodes at location 1 and 4: the blue route corresponds to option 1) and the green route corresponds to option 2).

We transform graph G into graph $G^{\mathcal{I}}$ by clustering the electricity- and gas nodes at the same location into a single location node. This is illustrated in Figure 3.7. Through clustering, we have defined a fully connected network graph $G^{\mathcal{I}}(V^{\mathcal{I}}, E^{\mathcal{I}})$ with $3n+1$ nodes. One of these nodes represents root node r, which has demand $b_r^g = b_r^e = 0$. The other $3n$ nodes represent the locations $i \in L$. Since the gas demand for each location is equal to zero, we find that the total demand of each location i is equal to b_i^e. Let E^S denote the set of edges from r to all the other nodes ("supply edges"), and let E^N denote the set of edges between any other pair of nodes ("network edges"). Then $E^{\mathcal{I}} = E^S \cup E^N$. Increasing capacity on the supply edges corresponds to building electricity supply units and increasing capacity on the network edges corresponds to building network connections between the different locations. We assume that the conversion rates are equal to one and distance loss factors are equal to zero, i.e. $\mu_{ij} = 1$ for each edge $(i, j) \in E^{\mathcal{I}}$.

In this instance of the MESDP, there are two different electricity supply types (PV and wind)

that can be built on the supply edges E^S and two different network connections that can be built on the network edges E^N (gas[3] and electricity), i.e., for each edge $(i,j) \in E^{\mathcal{J}}$, there are two different "facilities" that can be installed. Even though the facilities on supply or network edges are not of the same kind in real life, we model them as such by assuming that both the "supply facilities" and the "network facilities" consist of the same two sets: on all of the edges, we can install facilities with capacity equal to 1 (low-capacity (LC) facilities) and facilities with capacity equal to $B + 3M$ (high-capacity (HC) facilities). So, the LC facilities represent a wind supply unit on the supply edges and a gas connection on the network edges, and the HC facilities represent PV and an electricity line on the supply and network edges respectively. The capacity of a wind supply unit and a gas connection (LC facilities) are equal and the capacity of the PV units and the electricity connections (HC facilities) are equal. Furthermore, assume that the cost of installing a LC or HC facility on the supply edges is equal to 1 and the cost of installing a LC or HC facility on the network edges is equal to ϵ. In Table 3.2, the data for this instance is summarized.

Edge set	Asset to build	Modeled as	Cost	Capacity
Supply edges	PV	HC facilities	1	$B + 3M$
	Wind	LC facilities	1	1
Network edges	Electricity connections	HC facilities	ϵ	$B + 3M$
	Gas connections	LC facilities	ϵ	1

Table 3.2: An example of modeling the different assets as LC and HC facilities for the MESDP

We have defined the following instance of the MESDP:

Minimize

$$\sum_{(i,j)\in E^S} (z_{ij}^{HC} + z_{ij}^{LC}) + \sum_{(i,j)\in E^N} \epsilon \cdot (z_{ij}^{HC} + z_{ij}^{LC}) \tag{3.3a}$$

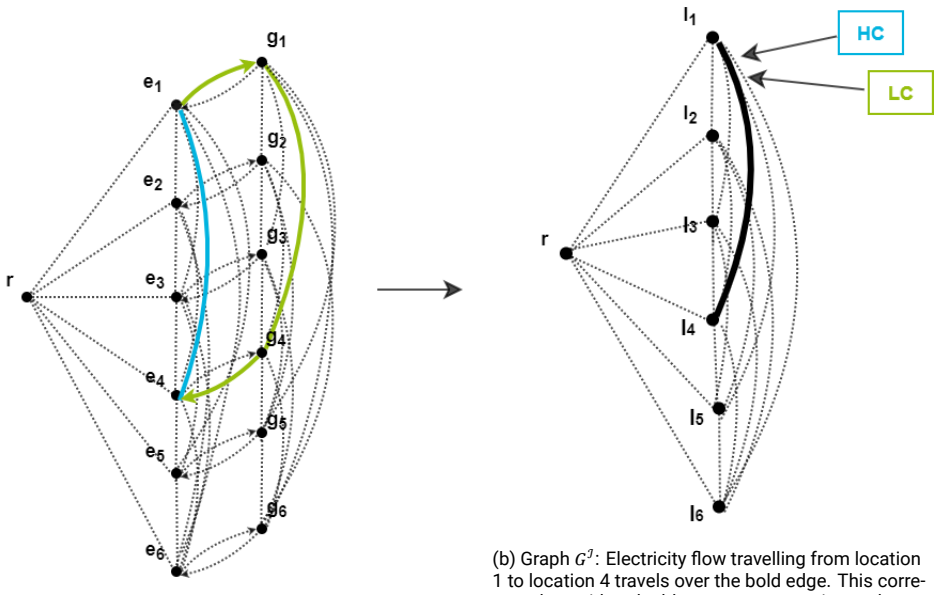
subject to

$$\sum_{\{j:(j,i)\in E^{\mathcal{J}}\}} x_{ji}^k - \sum_{\{j:(i,j)\in E^{\mathcal{J}}\}} x_{ij}^k = \begin{cases} -\sum_{\{j\in V: j\neq i\}} (a_j + M), & \text{if } i = r \\ a_i + M, & \text{if } i \neq r \text{ and } k = \text{electricity} \\ 0, & \text{if } i \neq r \text{ and } k = \text{gas} \end{cases} \tag{3.3b}$$

$$x_{ij}^k \leq z_{ij}^{LC} + (B + 3M) \cdot z_{ij}^{HC}, \quad \forall (i,j) \in E^{\mathcal{J}} \tag{3.3c}$$

$$z_{ij}^{LC}, z_{ij}^{HC} \in \mathbb{Z}^+ \text{ and } x_{ij}^k \in \mathbb{R}^+ \tag{3.3d}$$

[3] In this instance, a "gas connection" actually consists of three assets: a gas pipeline and two conversion units, for gas- electricity (CHP) and electricity-gas (P2G). This corresponds to the green route in Figure 3.7a. We model these three assets as a single gas connection, i.e., as a single facility in graph $G^{\mathcal{J}}$.



(a) Graph G: Flow travelling from electricity nodes at location 1 and 4 corresponds to the flow travelling over either the blue or green route.

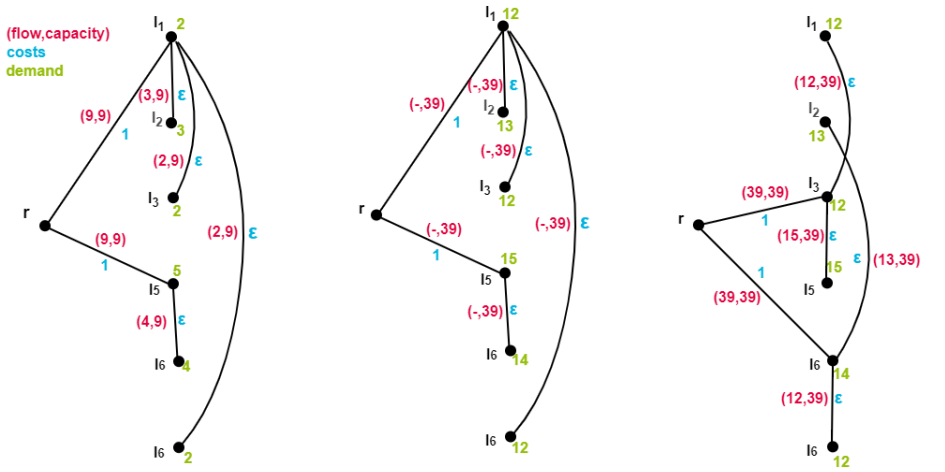
(b) Graph $G^{\mathcal{I}}$: Electricity flow travelling from location 1 to location 4 travels over the bold edge. This corresponds to either the blue or green route in graph G. The assets that have to be installed for the blue (red) route in G correspond to installing an HC (LC) facility on the bold edge in $G^{\mathcal{I}}$.

Figure 3.7: We can transform graph G into graph $G^{\mathcal{I}}$ by clustering the gas- and electricity nodes at the same location

An optimal solution to this instance of the MESDP has the following properties:

- An optimal design does not use LC facilities. The reason for this is that on any edge, installing a HC facility instead of a LC facility increases the capacity of the edge without increasing the cost.
- A feasible design has at least n HC facilities on the supply arcs E^S. This is because the total demand is $Bn + 3Mn$ and each HC facility has a capacity of $B + 3M$.
- The cost of an optimal solution is at least $n + 2n\epsilon$. Any solution with this cost has n HC facilities on the supply edges and $2n$ HC facilities on the network edges. An optimal design cannot place two or more HC facilities on one supply edge: if this would be the case, then more than $2n$ network edges contain a HC facility, and the total cost would then exceed $n + 2n\epsilon$. In Figure 3.9, this is illustrated. Therefore, we can assume that an optimal design places maximally a single HC facility on each supply edge.
- Since M is chosen to be sufficiently large, each node will be a transshipment node for at most two other nodes in an optimal solution with cost $n + 2n\epsilon$. In Figure 3.8 we illustrate why the M-term is necessary for this claim.

We claim that we have a Yes instance of 3-PART if and only if the optimal solution to the constructed instance $\mathcal{I}_{MESDP}$ of the MESDP has cost $n + 2n\epsilon$. To show the first direction of this claim, assume that we have a Yes instance to 3-PART, and that the partitions are given by $S_j = \{j_1, j_2, j_3\}$ for $1 \leq j \leq n$. Then, a Yes instance of the MESDP can be obtained by installing HC facilities on arcs $\{0, j_1\}$, $\{j_1, j_2\}$ and $\{j_1, j_3\}$ for $1 \leq j \leq n$.



(a) The optimal design when $M = 0$

(b) This design is not feasible when $M = 10$

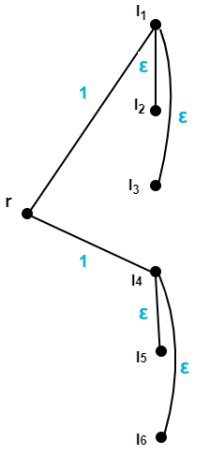
(c) An optimal design when $M = 10$

Figure 3.8: Take the following instance of 3-PART:

$$n = 2, \quad B = 9, \quad a_1 = 4, \ a_2 = 2, \ a_3 = 3, \ a_4 = 2, \ a_5 = 5, \ a_6 = 2$$

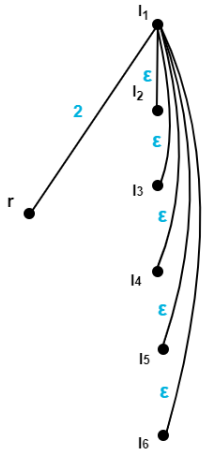
and let $M = 0$. Then there exists an optimal design for $\mathcal{I}_{MESDP}$ with cost $n + 2n\epsilon$, where node l_1 serves as transmission node for three other nodes (Figure 3.8a). This design does not correspond to a 3-partition of $a_1, \ldots, a_6$. In Figure 3.8b, we have set $M = 10$ for the same problem, and find that the design of Figure 3.8a is no longer feasible. In Figure 3.8c an optimal design for the case where $M = 10$ is given. This design does correspond with a 3-partition of $a_1, \ldots a_6$.

For proving the reverse direction of the claim, assume that we have a solution to the MESDP with cost $n + 2n\epsilon$. Then, we have used n facilities on the supply edges and $2n$ facilities on the network edges. The demand is satisfied for n nodes directly, and for $2n$ nodes through some transshipment node. As previously stated, the optimal design is a tree with $3n$ edges, where each node serves as a transshipment node for at most two other nodes. But then, we have a 3-partition of the nodes, where each partition S_j is given by a node j served directly by r and the two nodes for which j serves as a transshipment nodes. The total demand is $nB + 3nM$, which implies that each of the partitions S_j has a total demand of exactly $B + 3M$. Therefore, we have a Yes instance to the 3-PART problem. □



2 +4ε

(a) A design with a single HC facility on each supply edge and its corresponding costs



2 +5ε

(b) A design with two HC facilities on a single supply edge and its corresponding costs

Figure 3.9: Loading 2 HC facilities on a supply edge does not correspond with an optimal design.

4

Empirical Research

By highlighting model characteristics of the MESDP and investigating its computational complexity in Chapter 3 , we tried to gain insight into its difficulty. However, for investigating how difficult it is to solve practical problem instances, we cannot rely on reasoning and theory only. As the proof of the pudding is in the eating, empirical testing of algorithms is crucial for assessing their performance (Ahuja et al., 1993). In this chapter, we introduce various experiments with practical instances of the multi-energy system design problem that provide insights in the performance of MIP solvers.

4.1. Background

4.1.1. Theoretical vs. empirical research

In Chapter 3, we have proven that the MESDP is strongly NP-hard. Within the framework of computational complexity, we can now state that we are dealing with a "difficult" problem: it is not possible to find a polynomial-time algorithm to solve this problem, unless P = NP. However, the theory of computational complexity deals with *worst-case* running time of algorithms, i.e., the maximum number of iterations $T(n)$ that an algorithm needs for *any* input of size n. Predicting real-case running times based on worst-case estimates might fail, as these estimates could correspond to problem instances that are highly unlikely in practice (Bertsekas, 1998). To understand and predict the behaviour of algorithms for real-life instances of the MESDP, we therefore require empirical evidence. The empirical behaviour of an algorithm is often much better than its worst-case analysis suggests. A well-established example of this is the simplex method, that in its worst case does not run in polynomial time, but has proven to be very effective for solving problems in practice (Karp, 1975).

The statement above by no means implies that analyzing the computational complexity of an optimization problem is not valuable. Firstly, this analysis is often necessary for pointing out computational "bottlenecks" of many algorithms. By showing the strong NP-hardness of an instance of the MESDP with a single time period and two energy carriers, we found that solving a case with the possibility of loading different "facilities" on the edges is significantly more difficult than a case in which only a single energy carrier is considered. Therefore, we can expect that difficult trade-offs between different facilities could be a bottleneck for a solver. Secondly,

specifically for network flow problems, if we can point out the NP-hardness of a given problem (assuming that P ≠ NP), we can (and should) give up hope of formulating it as a regular minimum cost flow problem, which we know can be solved using polynomial algorithms (Bertsekas, 1998).

4.1.2. Assessing implementations

Before providing details on our experiments, we provide some background information on assessing implementations in an empirical study.

According to Ahuja et al. (1993), a typical empirical study tests multiple algorithms and consists of the following steps:

- Write a program for each algorithm to be tested;
- Generate different (random) problem instances with different input data;
- Run the implementations and compare their *effectiveness*.

For discussing the assessment of implementations, we make use of the framework as proposed by Maher et al. (2019). According to this framework, the following steps should be taken to compare the effectiveness of implementations:

- Measure the effectiveness of an implementation on a set of instances individually;
- Produce, for each implementation, summary statistics or a visualization that enables conclusions to be drawn about effectiveness across an entire class of instances.

The latter allows for identifying the “best” algorithm. General conclusions regarding this should preferably be done using statistical methods, but this is often difficult to do: since the true probability distributions of measures of effectiveness are unknown, they should be approximated by determining the empirical cumulative distribution function (CDF) with respect to a (small) set of test instances that are chosen to represent to full class. At best, this is a rough approximation, where a uniform probability distribution is often assumed for simplicity.

The aspect of effectiveness that we focus on is efficiency. A *measure of efficiency* for a given computation is the amount of a chosen resource that is required to perform that computation. Specifically for branch and bound implementations, which we investigate in this research, the efficiency of an implementation can be assessed by measuring its *resource consumption*, the *performed work* or the *progress* (Maher et al., 2019).

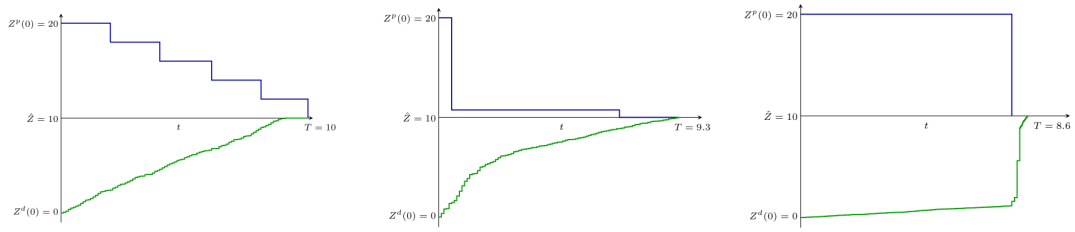
Measuring resource consumption The easiest way to assess the efficiency of an implementation is directly measuring the amount of resources required to achieve a given termination criterium. Typically, this resource is time (either CPU or wall clock). In branch and bound implementations, the *time to provable optimality* is the most common measure. For instances whose computation does not complete within a reasonable amount of time, the time to *a fixed optimality gap* is usually measured. And lastly, if feasible solutions are difficult to be found, the *time to the first solution* can be an appropriate measure. Some argue that it is debatable

whether time is the best resource measure to use (Ahuja et al., 1993): firstly, because times are often difficult to replicate due to the multiple sources of variabilities from the operating system itself, and secondly, being a measure that combines multiple measures of empirical performance, time does not provide detailed insight into the behaviour of an algorithm.

Measuring work performed Measures of work count the number of operations that are required to complete a computation, which is in many cases linearly related to the amount of time required. The first common measure of performed work in branch and bound implementations is the *number of nodes* explored in the tree for solving a problem to optimality. This measure does have a drawback, since it relies on the assumption that nodes have equal processing times, which is not always the case: different component algorithms of the MIP solver are executed at different nodes throughout the tree. When using the number of nodes as a measure of efficiency, it is necessary to understand which algorithms are executed during node processing. For example, generating more cuts at each node in the tree increases the process time of a node, but could reduce the total number of nodes in the tree. To measure the amount of work performed in processing all of the nodes, the *total number of iterations performed* over all nodes can be a good measure. Therefore, measuring the combination of the total number of processed nodes and the total number of performed iterations gives a more complete picture of the overall efficiency of an algorithm.

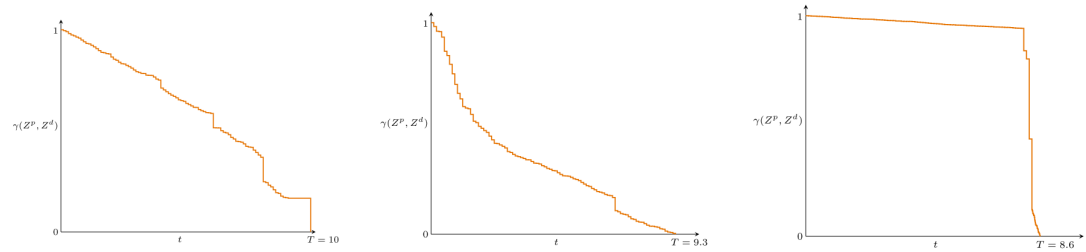
Measuring progress Measures of progress are important to provide insight in computations that cannot fully be completed within a given time limit. Instead of measuring resource consumption, it is measured how much "useful computation" can be done with a fixed amount of resources. Unfortunately, it is difficult to measure this in a rigorous way for branch and bound implementations: solving NP-hard problems can involve a significant amount of backtracking, which results in little perceivable progress (besides the elimination of dead ends). Despite this, it is essential to report on measures of progress, as it is not always possible to perform the full computation. Beside that, it provides insight in the time it takes to do a fixed fraction of the computation and allows us to determine the rate of progress for different resource settings. The most commonly used measure of progress is the *optimality gap*, which is calculated as in Equation (2.5). In Figure 4.1, three different examples of the evolution of the lower bound (the green curve) and incumbent (the blue curve) in implementations are given. Figure 4.2 shows the evolution of their corresponding optimality gaps. Figure 4.1a illustrates a constant and regular improvement of both the incumbent and the lower bound, resulting in the rather smooth gap function in Figure 4.2a. This is typically not the case: more often the biggest changes in the size of the gap come from large improvements in the incumbent value (with long periods of no improvements), while the lower bound has a more smooth evolution, as illustrated in Figure 4.1b and 4.1c. The difficulty of using the optimality gap as an accurate measure of progress is illustrated by these examples: after 5 seconds of computation, it appears that the implementation of Figure 4.1a outperformes the other two. However, this implementation has the largest total runtime and is therefore not necessarily the "best" one. Still, to our best knowledge, the optimality gap is an appropriate measure for comparing the progress made by different algorithms with the same resources. Caution should be taken for drawing

conclusions based on findings regarding the optimality gap, as large differences are not always very significant in this case.



(a) Implementation 1 - Bounds (b) Implementation 2 - Bounds (c) Implementation 3 - Bounds

Figure 4.1: Example graphs of incumbent and lower bound values in implementations (Maher et al., 2019)



(a) Implementation 1 - Gap (b) Implementation 2 - Gap (c) Implementation 3 - Gap

Figure 4.2: The evolution of the gaps corresponding to the graphs of Figure 4.1 (Maher et al., 2019)

In some cases, practitioners prefer implementations that take a long time to solve to optimality, but quickly find a good solution over implementations that require less time for optimality, but do not find good solutions early in the solution stage. To illustrate this, the implementation of Figure 4.2c might be the "winner" when we investigate the time to provable optimality or the time to a very small fixed gap. But this conclusion discards the total evolution of the gap function, something that can be very insightful when measuring progress. The *gap integral* (GI) is proposed as an alternative to the gap itself as a measure of progress (Maher et al., 2019). The gap integral measures the "average gap" over a given time interval by capturing the evolution of the gap over a limited computation time. Formally, it is defined as the following integral

$$GI = \int_0^T \gamma(Z^L(t), Z^U(t))\mathrm{d}t, \tag{4.1}$$

where T denotes the time needed to solve to optimality or the time limit and $\gamma(Z^L, Z^U)$ denotes the optimality gap for lower bound Z^L and current incumbent Z^U. When discrete samples of the upper and lower bounds are obtained, as often the case in practice, we can approximate the gap integral with the following formula.

$$GI = \sum_{i=1}^{I} \gamma(Z^L(t_i), Z^U(t_i)) \cdot (t_i - t_{i-1}), \tag{4.2}$$

where $t_i \in [0, T]$ and $i \in 1, \dots I$ are the discrete times at which the bounds are updated with $t_0 = 0$ and $t_I = T$. The GI is a number between zero and T, where numbers closer to zero indicate a more effective implementation. Revisiting the examples of gap functions in Figure 4.2, we can state that the second implementation (Figure 4.2b) has the best "average gap" over time and would therefore have the lowest GI. Even though it produces a single summary statistic, the GI value does capture the progress of an algorithm over time and is therefore an insightful measure.

4.1.3. MIP solvers

Practical mixed integer (linear) programs are typically solved by state-of-the-art MIP solvers (Klotz and Newman, 2013). Recent advantages in the implementation of algorithms and hardware improvements in MIP solvers allow practicioners to formulate and solve increasingly large and complex models. A MIP solver reads a MIP problem, executes an optimization procedure, containing both exact and heuristic methods, and returns the best solution found. For obtaining solutions to instances of the MESDP, we make use of Gurobi, a mathematical programming solver for mixed-integer programs. The methods that Gurobi applies for solving MIPs are a branch and bound procedure combined with problem reductions, cutting planes and various heuristic methods. More details on some of these components can be found in Chapter 2.3. We shortly introduce how these components are implemented in the solver.

Presolve The first step in solving a MIP is the presolve section. MIP presolve is a collection of problem reductions that reduce the size of the problem and tighten its formulation, i.e., the degree to which the constraints of the formulation accurately describe the underlying polyhedron of the solution space.

Branch and bound In the next step, the presolved problem is passed on to the branch and bound part of the solver. In the root node of the branch and bound tree, the LP relaxation is solved. After that, a branch and bound tree search is generated by iteratively selecting nodes as the next subproblem to process. The solver keeps track of the incumbent, the global lower bound and the gap between the incumbent and the lower bound throughout the whole procedure (optimality gap). When this gap is equal to 0, optimality is reached and the solver terminates.

Cutting planes The implementation of cutting planes is the most important contributor to computational advances in integer programming over the last years. During the branch and bound procedure, different cutting planes are applied to cut off relaxed solutions. Even though the set of automatically generated cutting planes is very powerful and robust, it can be interesting to vary the "aggressiveness" of the cut settings in some cases, especially with model-specific knowledge.

Heuristics The branching part of the branch and bound algorithm is not the only method that the solver uses to find new feasible solutions. Gurobi includes multiple heuristics, such as feasibility heuristics, local search heuristics and some additional heuristics in the root node. Good heuristic methods find solutions earlier than branching. By exploiting the problem struc-

ture, Gurobi adapts its strategy deciding when to apply which heuristics.

Additional techniques Besides the above mentioned components, many additional techniques are included in the solver's optimization procedure, such as sophisticated branch variable techniques, node presolve, symmetry detection, disjoint subtree detection, etc. In most cases, their goal is to limit the size of the branch and bound tree that must be explored.

Parameter tuning The behaviour of the methods that the solver applies can be controlled by adjusting their settings, i.e., tuning certain parameters. While default settings generally work well for solving MIPs, models can benefit from parameter tuning in some cases. By running various problem instances with the solver's default settings, practitioners can get more insight in possible problems for the solver. Based on these and on in-depth model knowledge, they can choose to experiment with different parameters and investigate whether the model benefits from this. Ideally, it should thoroughly be investigated whether (and how) the solver's efficiency can be enhanced through parameter tuning before turning to performance-enhancing strategies that require (much) more work, such as model reformulations or decomposition methods. There are a few reasons for this. First, from a practical point of view, this is a simple approach that might result in better computing times but hardly requires any effort to implement. Second, from a more theoretical point of view, it can provide researchers with additional insight in the solver's "bag of tricks" and the many underlying computational trade-offs that are a part of it. Third, as well as a better understanding of the solver itself, it provides additional insight in the model's characteristics and provides a direction for future research in performance enhancements.

An overview of the building blocks in the optimization procedure for MIPs is given in Figure 4.3. It should be clear that each box represents *"a giant bag of tricks"* (Gurobi Optimization Inc., 2017). As a detailed overview and explanation of these "tricks" lies beyond the scope of this thesis, we refer to Gurobi's reference manual (Gurobi Optimization Inc., 2017). The development of many of these tricks have proven to be very helpful for efficiently solving MIPs, and recent developments in solvers allow us to now solve very large and complex problems. However, they also make it rather difficult to fully understand the solver and its behaviour.

It is challenging to relate theoretical to empirical evidence when we make use of state-of-the-art mathematical programming solvers. Nevertheless, as there is a particular interest in solving practical instances of the MESDP, we gather empirical evidence using Gurobi.

4.2. Experiments

By assessing implementations of different problem instances of the MESDP, we intend to:

1. *find out which parts of the model are difficult for the solver by running various instances with default settings;*
2. *find out which parameter settings are potentially beneficial for the model by comparing implementations with different settings, where these settings are based on information gathered in the previous step*.

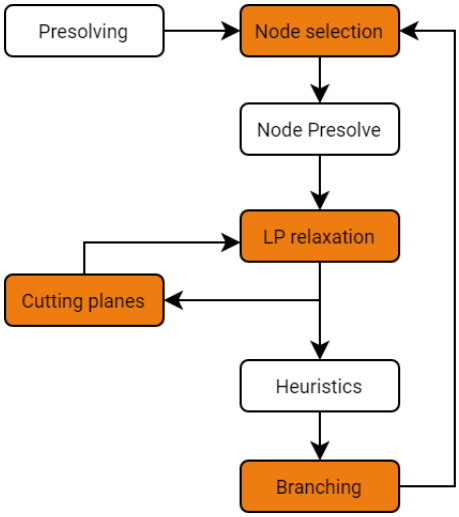


Figure 4.3: MIP building blocks used in Gurobi's optimization procedure (Gurobi Optimization Inc., 2017). The orange blocks represent the components that we assess in this empirical study.

It should be obvious that the efficiency of an implementation cannot be assessed by looking at the result of a single instance. According to Maher et al. (2019), randomly generated instances are usually not representative. The set of instance variations should rather be diverse with respect to particular properties of instances that seem complicating. This can however be difficult to achieve: computational experiments are a good way of highlighting these properties, but without a set of test instances, it is not possible to run computational experiments. We construct a set of test instances for the MESDP by varying the input data of the regular case. To construct this set, we perform various experiments with 1) smaller instances of the regular case and 2) variations on the MESDP that exclude constraints that we expect to be complicating. Details of the implementation and results of these experiments can be found in Chapter 7.1. Though these experiments, we have generated a set of instances for the MESDP for the remainder of this study. An overview of these is provided in Table 7.1.

4.2.1. Experiments with default settings

In this empirical experiment we intend to:

1. *find out which parts of the model are difficult for the solver by running various instances with default settings.*

We are interested in identifying difficult parts of the model, i.e., causes of performance problems for the solver, as we eventually want to to improve the solver's performance by optimizing its parameter settings. We assess the performance of the solver on this set of instances, based on the following various measures of efficiency.

Time to optimality The first measure of efficiency that we use for assessing the quality of the solver for various instances is the time to optimality. To this end, we should let the solver run without a time limit and examine how much time is required for solving the problem optimally.

Solving the relaxation The continuous model that gets solved at the root node of the MILP model can be solved with different algorithms: the primal simplex method, the dual simplex method or the barrier method (also known as interior point algorithms) (Gurobi Optimization Inc., 2019). When deciding which algorithm suits the model best, it is important to investigate the *degeneracy* of the model. If the model is degenerate, i.e., if it has a basis where a basic variable is equal to zero, the simplex method is not guaranteed to be finite (Orlin and Nasrabadi, 2013). A problem that is primal degenerate might work well with a dual simplex method, and vice versa. A problem that is both primal and dual degenerate, might work well with the barrier method. The MESDP is highly degenerate, as many investment decision variables will have a value of zero in the optimal solution. For this reason, we conduct various experiments with all of the three methods on the set of instances.

Time to solve to a fixed gap To get more insight in the time the solver requires to come up with good quality solutions, we let the solver run until a solution with a fixed optimality gap is found.

For the results of the above experiments, we refer to Chapter 7.2. We would like to highlight an important finding from these experiments here however, as our strategy in the remainder of the empirical study is based on these initial results. In almost all cases, after a while there is very little progress in the best bound and no progress at all in the incumbent, resulting in an optimality gap that decreases extremely slowly. In smaller cases that were solved to optimality, the optimal solution was found in an early stage, while the lower bound kept increasing very slowly. This particular result can not necessarily be extended to the larger and more complicated cases: without knowledge on the optimal objective function one can not state whether the incumbent will not change anymore throughout the remainder of the solution process. Still, this result for the incumbent values in the smaller cases do motivate us to investigate ways in which we can improve the convergence of the lower bound; considering the slowly improving lower bound function that we also see in the larger cases.

4.2.2. Experiments with parameter tuning

In this empirical experiment we intend to:

2. *find out which parameter settings are potentially beneficial for the model by comparing implementations with different settings, where these settings are based on information gathered in the previous step,*

where the slowly improving lower bound function is the information on which we should base the settings.

Klotz and Newman (2013) propose guidelines for the performance bottleneck of a lack of progress in the best bound. To strenghten the solver's performance, they propose to adjust different parameters, as this is easily done and might have a significant effect.

We investigate the effect of changing the following parameters for all of the instances in the test set:

- Apply aggressive cut generation (for all cutting plane methods)
- Apply aggressive model-specific cuts
- Apply strong branching
- Let the solver focus on finding the best bound in node selection

In Figure 4.3 it is illustrated in which components of the solver the parameter adjustments are made.

Aggressive cut generation Aggressive cut generation allows for the solver to spend more time calculating cutting planes throughout the entire algorithm. This results in more cuts, and thus a further tightening of the formulation, but might also result in a slower rate of node processing due to the additional constraints in the node LPs.

Aggressive model-specific cut generation The cuts that are added by the solver rely on general polyhedral theory that applies MIPs. It is less likely that a state-of-the-art MIP solver implements cuts that rely on a structure that is specific to indivual MIPs (Klotz and Newman, 2013). Cuts that have proven to be particularly useful for fixed charge network flow problems are flow cover cuts. MIP solvers have the ability to add these, but might not always recognize the network design structure of a model. Forcing the solver to spend a more effort in calculating flow cover cuts might therefore help its performance.

Strong branching When strong branching is applied, solvers run a (modest) number of iterations on each branching variable candidate at each node, testing which one gives the best improvement before actually branching on them. Infeasible branches are exploited, tightening the problem formulation in the node. A trade-off has to be made, as strong branching does increase the computation at each node in the tree.

Focus on best bound in node selection Letting the solver select the node with the minimal relaxation objective value (while ignoring the number of integer infeasibilities) allows for it to update the best node value faster. However, this also may cause the solver to find less integer feasible solutions. In our case, the solver finds feasible integer solutions rather quickly, while the objective bound moves slowly. Best bound node selection could therefore help the solver's performance.

For the results of these experiments, we refer to Chapter 7.2.

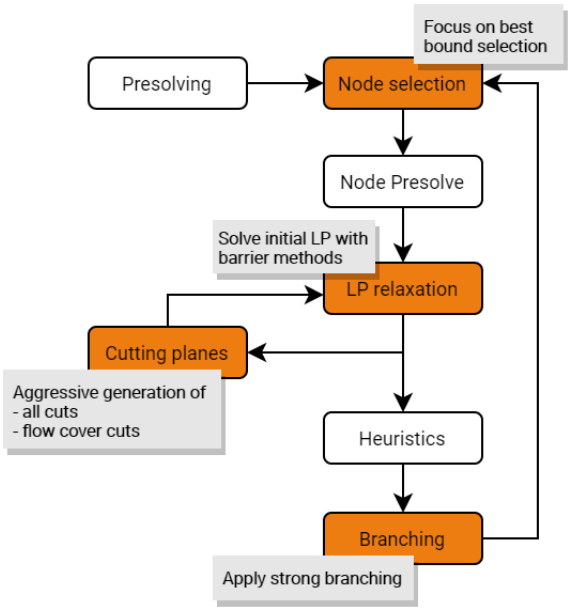


Figure 4.4: The parameters that we vary in this study with the components in which this is applied.

5

Valid inequalities

In this chapter, we investigate the influence of including valid inequalities on solving the MESDP.

5.1. Theory

Knowledge on model characteristics allows practitioners to formulate their own cuts in order to improve the solver's performance. There is a large amount of theory on cut derivation for MIPs available. And even though in-depth knowledge of this adds to the possibility of improving the solver's performance, there are some fairly simple techniques for deriving useful cuts (or *valid inequalities*) that have proven to enhance run times (Klotz and Newman, 2013).

Valid inequalities A *valid inequality* (VI) is any constraint that does not eliminate any feasible solutions to the original problem. These VI or *cutting planes* should eliminate part of the LP feasible region (Orlin and Nasrabadi, 2013). The *convex hull* is the smallest LP feasible region containing all of the integer solutions. For a MILP, it holds that solving an LP with the convex hull of integer solutions as feasible solution space guarantees finding the optimal solution, because all of the corner points are integer. Finding constraints of the convex hull or finding the convex hull itself would therefore result in better bounds. However, this is extremely difficult to do. More commonly in practice, useful VI are looked for, where useful implies that they cut of some fractional solutions, but no integer solutions.

For a while, it has been possible to add VI during the branch and bound process for most MIP solvers (CPLEX, for instance, introduced this already in 2006). A somewhat new development for MIP solvers is the introduction of *lazy constraints*.

Lazy constraints The amount of time required to solve an MIP can heavily depend on the number of constraints (Pearce and Hons, 2019). A *lazy constraint* is a constraint that is only considered by the solver once it is violated. Using lazy constraints allows practitioners to "leave out" constraints from a problem that have to be *satisfied* for any *feasible* solution, but which may not be *required* by the solver to find the *optimal* solution. Leaving out some of these constraints results in a smaller model. When solving this relaxed model, each integer solution found during the branch and bound procedure is checked. If this solution violates one of the constraints that is not included in the relaxed model, i.e., one of the lazy constraints, the inte-

ger solution is discarded and the violated constraint is pulled back into the active model. In this way, only the constraints that are required for finding the optimal solution are included. It can be useful to designate constraints as lazy when you know beforehand that they are trivially satisfied (or rarely violated) in feasible solutions, as this reduces the total number of constraints that the solver has to deal with.

Besides the ability to designate constraints as lazy before solving a model, lazy constraints can be implemented through the use of *callback functions*. Modern solvers have callback capabilities that allows the user to retrieve information and change parts of a model during the solution process. Using callback functions, cuts and lazy constraints can be added dynamically every time that a feasible solution is found in the branch and bound tree, while continuing the solving process. In this sense, lazy constraints might be considered user-generated "cutting planes" that *are* allowed to cut off feasible solutions. A graphical representation of this idea is given in Figure 5.1.

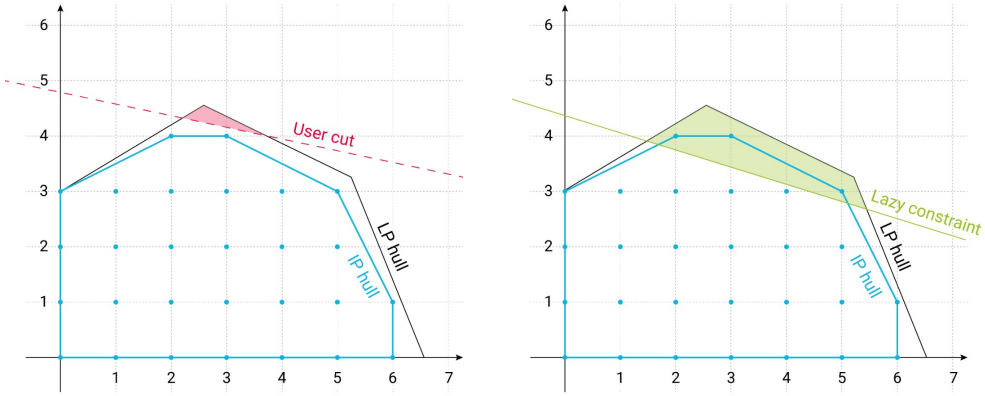


Figure 5.1: The difference between user cuts and lazy constraints for the convex hull of a MILP. Lazy constraints are, opposed to user cuts, allowed to cut off a part of the feasible region.

Pearce and Hons (2019) highlight three main scenarios in which lazy constraints are useful:

1. Models with an exponential-sized set of constraints;
2. Models with a large set of redundant constraints;
3. In cases where not all constraints can be identified at the model initialization phase, for instance within the application of Benders decomposition.

Both lazy constraints and VI can iteratively be added through the use of callback functions. It is crucial to inform the solver whether the constraint you want to add from the callback function is an actual cutting plane or a lazy constraint: adding a lazy constraint as a cutting plane might cut of feasible solutions, something that a cutting plane should *never* do. Calling a constraint a cutting plane tells the solver that it cannot cut of a feasible solution, so the solver will not check the violation of such a cutting plane every time a feasible solution is found. This can lead to incorrect results. Lazy constraints on the other hand are guaranteed to be checked by the solver every time a feasible solution is found.

In the following section, we apply theory from VI to the MESDP.

5.2. Valid inequalities for the MESDP

In this section, we identify valid inequalities that tighten the formulation of the MESDP and investigate their influence on the solver's performance. Our choices of VI are motivated both by successful applications in similar problems in previous research and some computational experience regarding outcomes of solving the MESDP. From common situations that often arise in the solutions we can, rather intuitively, draw some patterns that allow us to predict which VI might have a positive effect.

Single node inequalities

Mass balance constraint (3.1a) states that in each time period $t \in T$ the net flow in each node $i \in V$ should be equal to its demand b_{it} in that time period. As stated before, inflow can come from 1) other nodes, through arcs (conversion units) or edges (network connections); 2) supply units at node i (only in electricity- or gas nodes) or 3) storage units at node i. Outflow can go to 1) other nodes or 2) storage units at node i.

$$\sum_{\{j:(j,i)\in E\}} \mu_{ji} \cdot x_{jit} - \sum_{\{j:(i,j)\in E\}} x_{ijt} + S_{it} - W_{it}^{IN} + W_{it}^{OUT} = b_{it}, \qquad \forall t \in T, \forall i \in V, \forall t \in T \quad (3.1a)$$

Since $x_{ijt}, W_{it}^{IN} \geq 0 \forall i \in V, (i,j) \in E, t \in T$, the following holds.

$$\sum_{\{j:(j,i)\in E\}} \mu_{ji} \cdot x_{jit} + S_{it} + W_{it}^{OUT} \geq b_{it}, \qquad \forall i \in V, \forall t \in T \quad (5.1)$$

Combining constraints (3.1k) and (3.1l), it follows that the amount of flow taken from storage at node $i \in V^k$ in time period t, W_{it}^{OUT}, cannot exceed the total installed capacity of storage in the previous time periods, $u^k \cdot \sum_{q=1}^{t-1} z_{iq}^W$. Therefore, taking into account loss and efficiency factors η_k and ξ_k, the following holds.[1]

$$W_{it}^{OUT} \leq \xi_k^2 \cdot \eta_k \cdot u^k \cdot \sum_{q=1}^{t-1} z_{iq}^W, \qquad \forall i \in \bigcup_{k\in K} V^k, \ \forall t \in T \quad (5.2)$$

The total amount of inflow from other edges, $\sum_{\{j:(j,i)\in E\}} \mu_{ji} \cdot x_{jit}$, depends on the total installed capacity on those edges, which is for each edge $(j,i) \in E$ equal to $u_{ji} \cdot \sum_{q=1}^{t} z_{jiq}$ (constraint (3.1d)). We obtain an upper bound for the total inflow in node i from other edges.

$$\sum_{\{j:(j,i)\in E\}} \mu_{ji} \cdot x_{jit} \leq \sum_{\{j:(j,i)\in E\}} \mu_{ji} \cdot u_{ji} \sum_{q=1}^{t} z_{jiq}, \quad \forall i \in V, \forall t \in T, \quad (5.3)$$

[1] Note that this inequality is not necessarily strong, as we only consider standing losses η_k from a single time period. This does allow us to take η_k outside of the sum, which is necessary for the way we derive the VI here.

from which we obtain

$$\sum_{\{j:(j,i)\in E\}} \mu_{ji} \cdot x_{jit} \leq \max_{(j,i)\in E}(\mu_{ji} \cdot u_{ji}) \cdot \sum_{\{j:(j,i)\in E\}} \sum_{q=1}^{t} z_{jiq}, \quad \forall i \in V, \forall t \in T. \tag{5.4}$$

For supply S_{it}, we have the following inequality (constraint (3.1b)).

$$S_{it} \leq \sum_{s} \sum_{q=1}^{t} (u_t^s \cdot z_{iq}^s) + g_{it}, \qquad \forall t \in T, \forall i \in V, \tag{5.5}$$

from which we derive

$$S_{it} \leq \max_{s}(u_t^s) \cdot \sum_{s} \sum_{q=1}^{t} z_{iq}^s + g_{it}, \qquad \forall t \in T, \forall i \in V. \tag{5.6}$$

Combining inequalities (5.1), (5.2), (5.4) and (5.6), we obtain the following inequality.

$$\max_{(j,i)\in E}(\mu_{ji} \cdot u_{ji}) \cdot \sum_{\{j:(j,i)\in E\}} \sum_{q=1}^{t} z_{jiq} + \max_{s}(u_t^s) \cdot \sum_{s} \sum_{q=1}^{t} z_{iq}^s + \xi_k^2 \cdot \eta_k \cdot u^k \cdot \sum_{q=1}^{t-1} z_{iq}^W \geq b_{it} - g_{it}, \qquad \forall i \in \bigcup_{k\in K} V^k, \ \forall t \in T.$$

As $z_{iq}^s = 0 \quad \forall i \in V^g \cup V^h$, we can strengthen this inequality for the gas- and heat nodes.

$$\max_{(j,i)\in E}(\mu_{ji} \cdot u_{ji}) \cdot \sum_{\{j:(j,i)\in E\}} \sum_{q=1}^{t} z_{jiq} + \xi_k^2 \cdot \eta_k \cdot u^k \cdot \sum_{q=1}^{t-1} z_{iq}^W \geq b_{it} - g_{it}, \quad \forall i \in V^g \cup V^h, \forall t \in T. \tag{5.7}$$

Let

$$A_{it} = \begin{cases} \max\left(\max_{(j,i)\in E}(\mu_{ji} \cdot u_{ji}), \ \max_s(u_t^s), \ \xi_k^2 \cdot \eta_k \cdot u^k \right), & \text{if } i \in V^e \\ \max\left(\max_{(j,i)\in E}(\mu_{ji} \cdot u_{ji}), \ \xi_k^2 \cdot \eta_k \cdot u^k \right), & \text{if } i \in V^g \cup V^h \end{cases}, \quad \forall t \in T.$$

We then find

$$A_{it} \cdot \Big(\sum_{\{j:(j,i)\in E\}} \sum_{q=1}^{t} z_{jiq} + \sum_{s} \sum_{q=1}^{t} z_{iq}^{s} + \sum_{q=1}^{t-1} z_{iq}^{W} \Big) \geq b_{it} - g_{it}, \quad \forall i \in \bigcup_{k\in K} V^k, \ \forall t \in T, \tag{5.8}$$

which is equivalent to

$$\sum_{\{j:(j,i)\in E\}} \sum_{q=1}^{t} z_{jiq} + \sum_{s} \sum_{q=1}^{t} z_{iq}^{s} + \sum_{q=1}^{t-1} z_{iq}^{W} \geq \frac{b_{it} - g_{it}}{A_{it}}, \quad \forall i \in \bigcup_{k\in K} V^k, \ \forall t \in T. \tag{5.9}$$

The left-hand side of Equation (5.9) is integer-valued, which is why we can state the following.

$$\sum_{\{j:(j,i)\in E\}} \sum_{q=1}^{t} z_{jiq} + \sum_{s} \sum_{q=1}^{t} z_{iq}^{s} + \sum_{q=1}^{t-1} z_{iq}^{W} \geq \left\lceil \frac{b_{it} - g_{it}}{A_{it}} \right\rceil, \quad \forall i \in \bigcup_{k\in K} V^k, \ \forall t \in T. \tag{5.10}$$

This is the first (globally valid) inequality that we derive for the MESDP.

Inequalities based on the graph structure

To derive VI from the graph structure of the MESDP, we re-introduce the time-expanded graph of G, $G^T = (V^T, E^T)$, where each node $i \in V^T$ represents a node from V in a distinct time period $t \in T$ or a gas/electricity supply node, and each arc/edge $(i,j) \in E^T$ represents i) the edges from E in the distinct time periods if i and j are nodes in the same time period; ii) storage, if i and j belong to different time periods; and iii) supply, if i is one of the supply nodes.

Considering the fact that a feasible flow in the time-expanded graph G^T corresponds to a feasible solution for the MESDP, we can derive VI by defining *graph cuts* in graph G^T - where we are motivated by the fact that facet defining inequalities for the general network loading problem have been derived using similar cut inequalities. The capacity of a cut in a classical network graph should be at least the sum of the requirements separated by this cut (Barahona, 1996). Due to the presence of arc multipliers in the MESDP, we cannot apply this result in network graph G^T. We can however apply a similar idea and derive (potentially weaker) cuts.

Consider Figure 5.2, 5.3, 5.4 and 5.5, illustrating cuts in graph G^T.

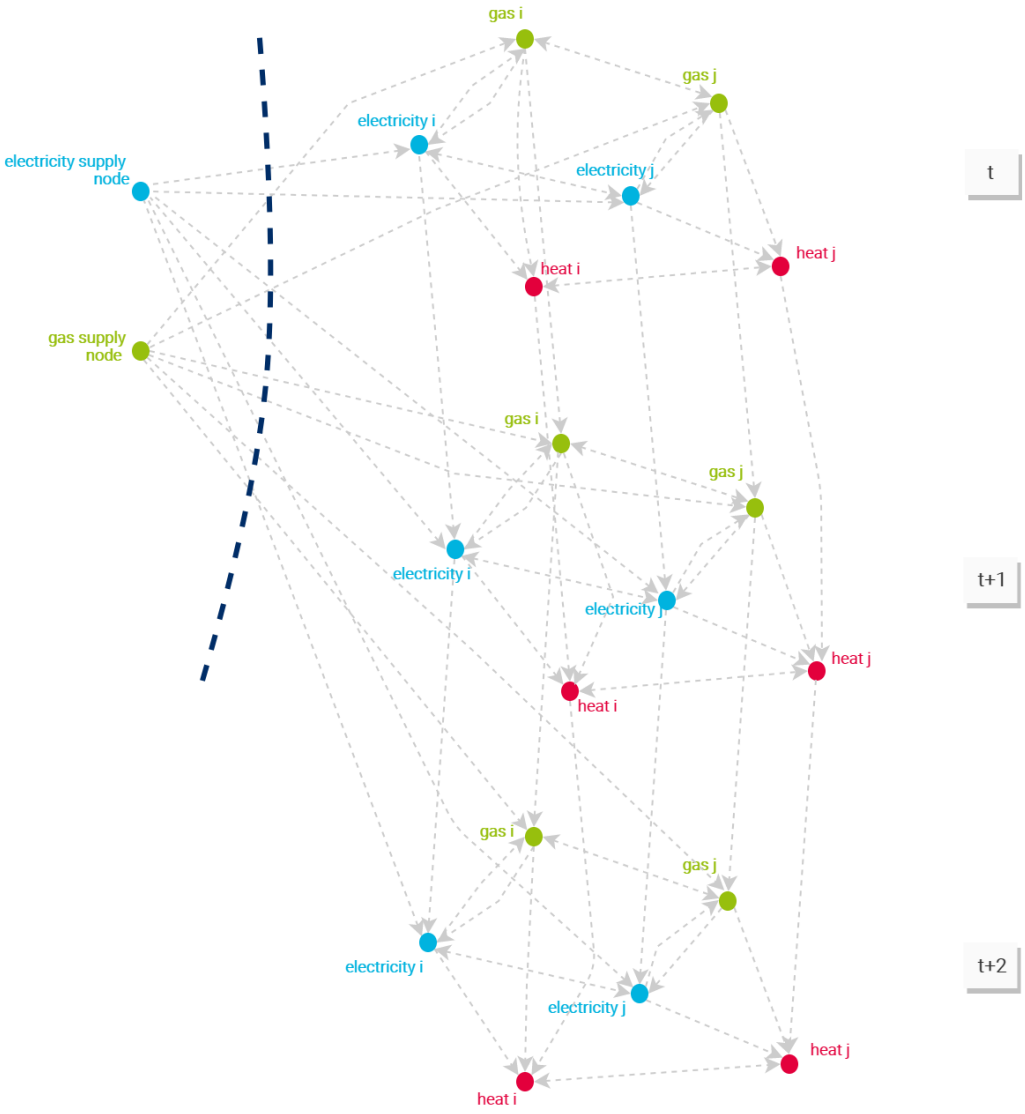


Figure 5.2: Cut-sets in the time-expanded graph G^T (1)

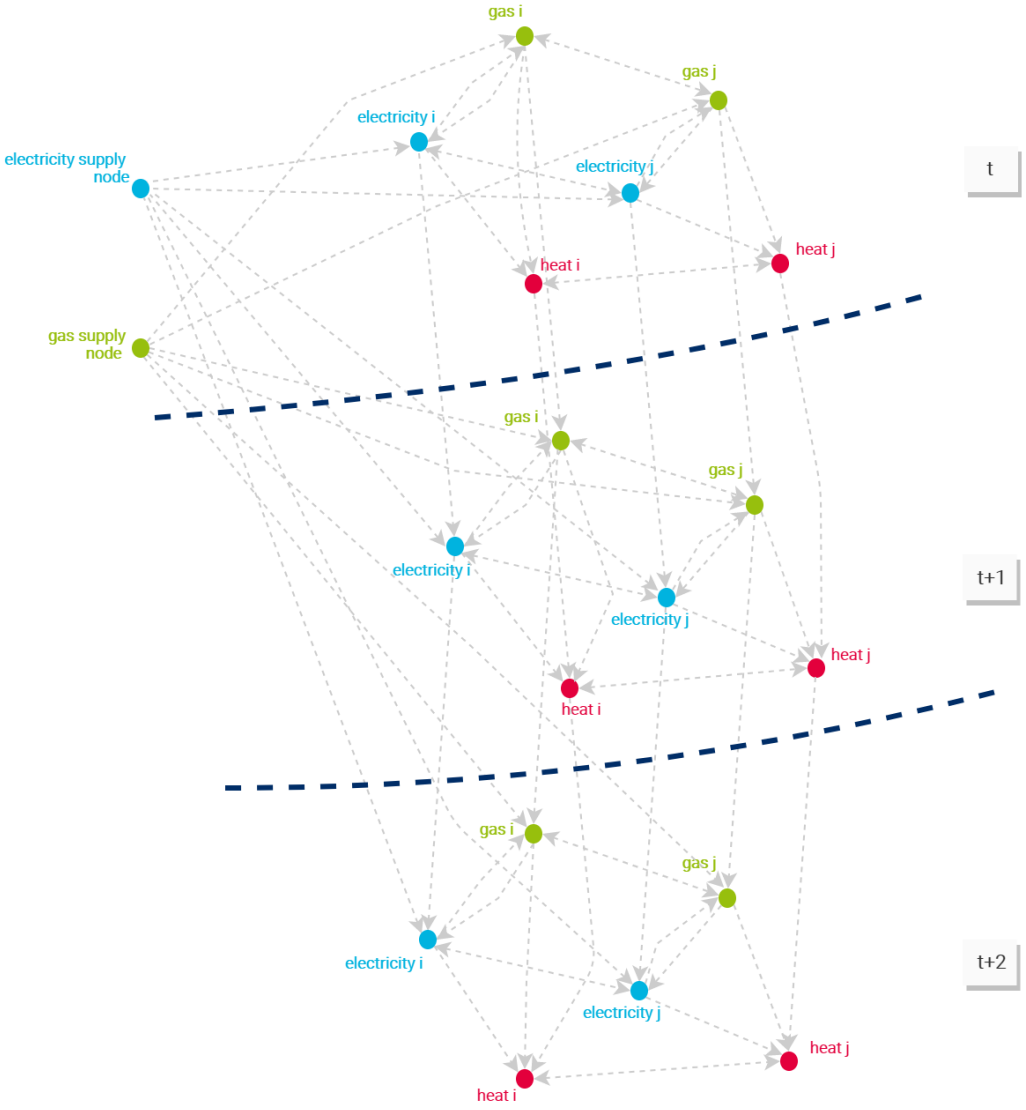


Figure 5.3: Cut-sets in the time-expanded graph G^T (2)

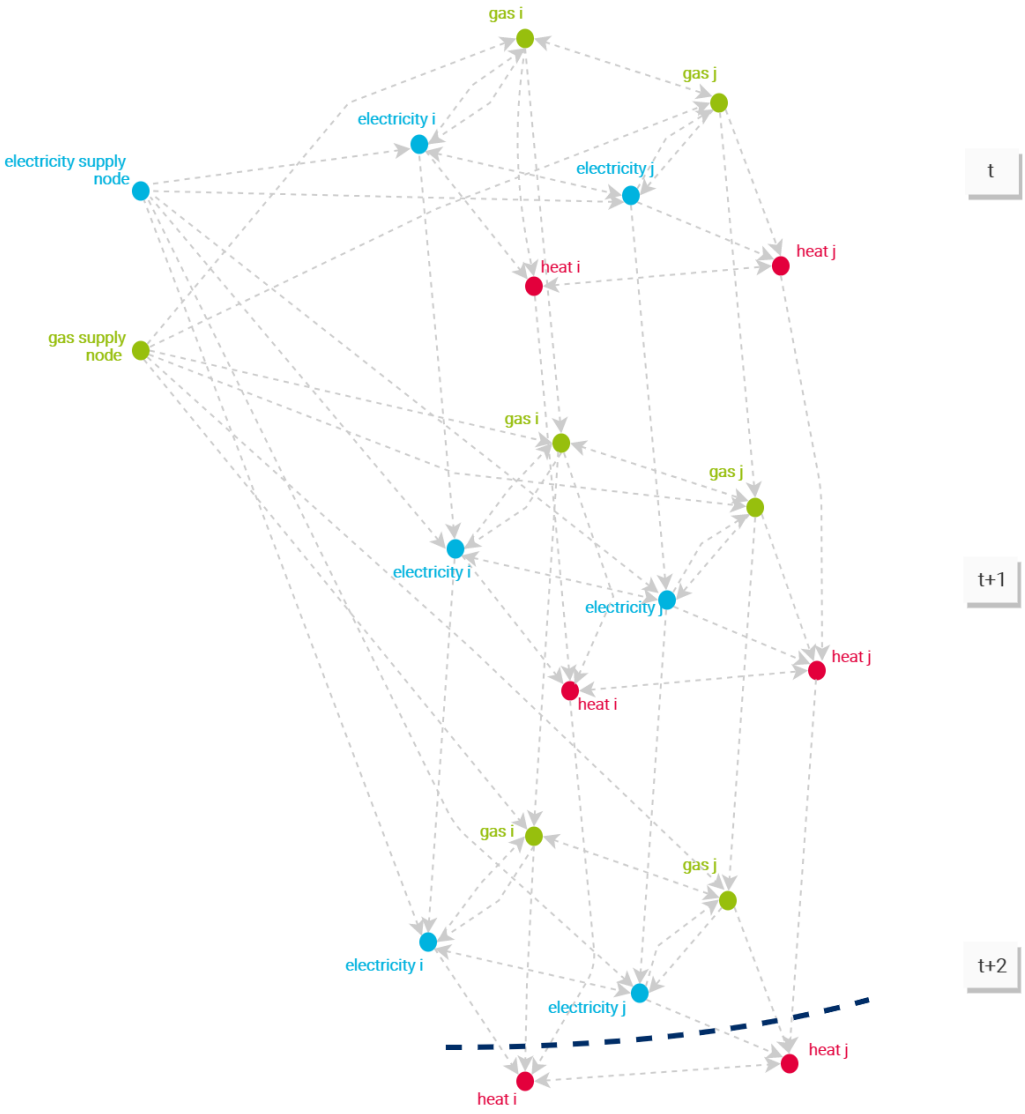


Figure 5.4: Cut-sets in the time-expanded graph G^T (3)

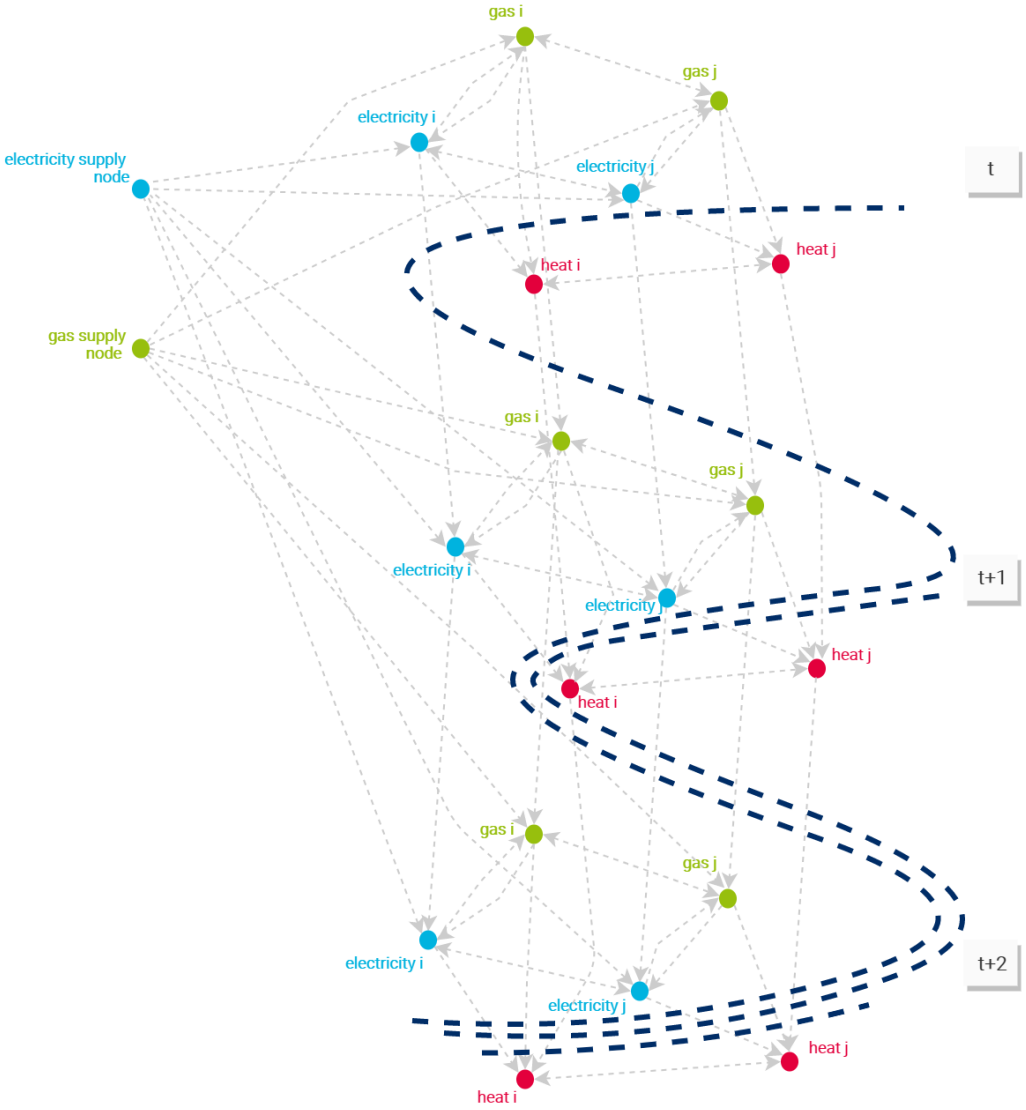


Figure 5.5: Cut-sets in the time-expanded graph G^T (4)

Due to the generalized flow in our problem, conservation of flow does not hold in graph G^T. This has some implications for the way that we derive VI from cuts in the graph: if for all edge multipliers we would have $\mu_{ij} \leq 1$, then we could state, for example for the cut in Figure 5.2, that the capacity of the cut, i.e., the maximum amount supplied by the electricity- and gas supply nodes, should equal at least the total demand over all time periods.

$$\sum_{i \in V} \sum_{t \in T} \left(\sum_{s} u_t^s \cdot \sum_{q=1}^{t} z_{iq}^s + g_{it} \right) \geq \sum_{i \in V} \sum_{t \in T} b_{it} \tag{5.11}$$

However, as flow is not conserved, Equation (5.11) does **not** hold for the MESDP: flow can be "created" on edges or arcs where $\mu_{ij} > 1$, implying that in some feasible solutions the capacity of the cut might be less than the total demand. Therefore, we should take edge multipliers μ_{ij} into account when deriving VI from cut-sets in the graph.

Let P_i be a possible path from the electricity- or gas supply node to a node i in the time-expanded graph V^T. Through storage, flow can travel from or to the next time period, which is why the storage arcs should also be taken into account in path P_i. Note that storage efficiency and loss factors therefore correspond to arc multipliers in the time-expanded graph. Let θ_{ij} denote such an arc multiplier for arc $(i,j) \in E^T$, i.e.,

$$\theta_{ij} = \begin{cases} \mu_{ij}, & \text{if } i,j \in V^t \\ \eta_k \cdot \xi_k^2, & \text{if } i \in (V^t)^k \text{ and } j \in (V^{t+1})^k. \end{cases}$$

For each unit flow leaving the electricity supply node and traveling over path P_i, $\prod_{(j,l) \in P_i} \theta_{jl}$ units of flow arrive at node i.

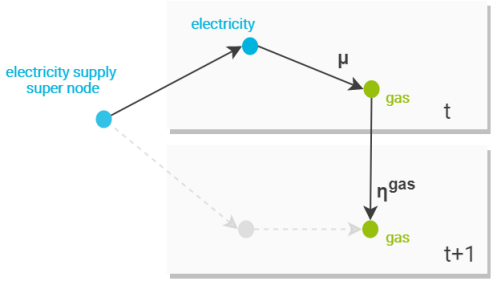


Figure 5.6: A possible flow path in the time-expanded graph for a single location.
For each unit flow that travels from the supply node to the gas node in $t+1$, $\mu \cdot \eta^g$ units of flow arrive [2].

For each $i \in V^T$, let $\mathrm{M}_i = \max_{P_i} \left(\prod_{(j,l) \in P_i} \theta_{jl} \right)$, i.e., the maximum amount of flow that can be created or the least amount of flow that is lost (depending on whether $\prod_{(j,l) \in P_i} \theta_{jl} < 1$) on each path P from the electricity supply node to node i. The path corresponding to M_i never contains any directed cycles (dicycles) in G^T: first and foremost, because there are no dicycles that create flow - the only arcs where $\mu_{ij} > 1$ are the (directed) HP arcs - and in addition, if a

[2] Where μ, in this case, equals the conversion rate of a P2G unit.

path P_i' contains a directed cycle C with $\prod_{(i,j)\in C} \theta_{ij} < 1$, then there exists a path $P_i'' = P_i' \backslash C$, where we have $\prod_{(i,j)\in P_i''} \theta_{ij} > \prod_{(i,j)\in P_i'} \theta_{ij}$.

Therefore, it holds that $\mathrm{M}_i > 0$ and finite for each $i \in V^T$.

We now arrive at the following result. Let $\sum_{i\in V}\sum_{t\in T} b_{it}$ denote the total demand for all nodes in all time periods. We know that the total amount of supplied flow from the electricity- and gas supply node, $\sum_{i\in V}\sum_{t\in T} S_{it}$, should suffice to fulfill the total demand $\sum_{i\in V}\sum_{t\in T}(b_{it})$, i.e., be greater or equal than the total amount of flow that arrives at the non-supply nodes in V^T (*while taking into account the gain- and loss factors*), implying that:

$$\sum_{i\in V}\sum_{t\in T} S_{it} \geq \sum_{i\in V}\sum_{t\in T} \frac{b_{it}}{\mathrm{M}_i} \tag{5.12}$$

Combining Equation (5.12) with capacity constraints (3.1b), we find the following.

$$\sum_{i\in V}\sum_{t\in T}\Big(\sum_{s} u_t^s \cdot \sum_{q=1}^{t} z_{iq}^s + g_{it}\Big) \geq \sum_{i\in V}\sum_{t\in T} \frac{b_{it}}{\mathrm{M}_i} \quad \Rightarrow \tag{5.13}$$

$$\max_{s,t}(u_t^s) \cdot \sum_{i\in V}\sum_{t\in T}\Big(\sum_{s}\sum_{q=1}^{t} z_{iq}^s + g_{it}\Big) \geq \sum_{i\in V}\sum_{t\in T} \frac{b_{it}}{\mathrm{M}_i} \quad \Rightarrow \tag{5.14}$$

$$\sum_{i\in V}\sum_{t\in T}\sum_{s}\sum_{q=1}^{t} z_{iq}^s \geq \frac{1}{\max_{s,t}(u_t^s)} \sum_{i\in V}\sum_{t\in T}\Big(\frac{b_{it}}{\mathrm{M}_i} - g_{it}\Big) \tag{5.15}$$

As the left-hand side of Equation (5.15) is integer, the following holds.

$$\sum_{i\in V}\sum_{t\in T}\sum_{s}\sum_{q=1}^{t} z_{iq}^s \geq \left\lceil \frac{1}{\max_{s,t}(u_t^s)} \sum_{i\in V}\sum_{t\in T}\Big(\frac{b_{it}}{\mathrm{M}_i} - g_{it}\Big) \right\rceil \tag{5.16}$$

The left-hand side of this (globally valid) inequality corresponds to the capacity of the cut-set in Figure 5.2.

To strenghten the VI in Equation (5.16), we consider the cuts in Figure 5.3. In addition to the cut corresponding to the total demand over all time periods, we apply cuts in which the total demand in **each** time period is considered. It is important to take the flow between time periods, i.e., from storage, into account here. Instead of dividing the sum of the demand by the maximum supply capacity as in (5.16), we should now also account for storage capacities. To this end, we define

$$B_t = \begin{cases} \max(\max_k(u^k), \max_s(u_t^s)), & \text{if t>1} \\ \max_s(u_t^s)), & \text{else} \end{cases}$$

and t_{max} is equal to the last time period, i.e., $t_{max} = \max_{t\in T}(t)$.

In a similar way as before, we arrive at the following (globally valid) inequalities:

$$\sum_{i\in V}\sum_{\tau=t}^{t_{max}}\sum_{s}\sum_{q=1}^{\tau} z_{iq}^{s} + \sum_{k\in K}\sum_{i\in V^k}\sum_{q=1}^{t-1} z_{iq}^{W} \geq \left\lceil \frac{1}{B_t} \cdot \sum_{i\in V}\sum_{\tau=t}^{t_{max}} \left(\frac{b_{i\tau}}{\mathrm{M}_i} - g_{i\tau}\right) \right\rceil, \qquad \forall t \in T \tag{5.17}$$

Lastly, we consider "terminal" nodes in G^T, i.e., nodes that cannot pass flow on to other nodes. As there are no actual terminal nodes in G^T (all nodes can pass flow on), we create one by aggregating the heat nodes in the last time period, as illustrated in Figure 5.7. Even though we can only consider a small part of the integer investment variables in this specific cut, we might still find this cut to be useful.

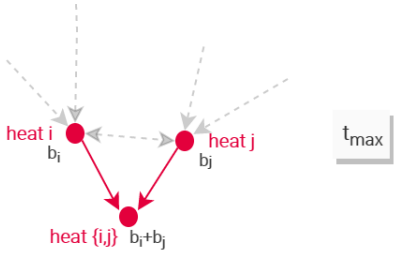


Figure 5.7: Aggregating the heat nodes in a single terminal node at $t = t_{max}$.

We derive a VI corresponding to the cut in Figure 5.4. In the figure, the capacity of the cut is equal to the total capacity of installed heat storage at $t + 1$, and the total capacity of installed CHP and HP units at $t + 2$, i.e., the total capacity of all edges from $(V^{t+2})^g$ and $(V^{t+2})^e$ to $(V^{t+2})^h$.

The total installed heat storage at a time t is equal to $u^h \cdot \sum_{i\in V^h}\sum_{q=1}^{t} z_{iq}^{W}$, and the total installed CHP and HP units at a time t is equal to $\sum_{i\in V^h}\sum_{\substack{\{j\in(V^g\cup V^e):\\(j,i)\in E\}}} u_{ji} \cdot \sum_{q=1}^{t} z_{jiq}$.

We can therefore derive the following inequality from the cut in Figure 5.4, where t_{max} denotes the last time period in T, and M_i^h denotes the maximum of the capacity of a CHP unit, HP unit, or heat storage unit, times their respective multipliers, i.e.,

$$\mathrm{M}_i^h = \max(\max\{(\mu_{ji} \cdot u_{ji}) |\ (j,i) \in E : j \in V^g \cup V^e\}, \xi_h^2 \cdot \eta_h \cdot u^h).$$

In other words, M_i^h is equal to the maximum amount of flow that can arrive at a heat node i from a single asset installed on either CHP, HP or heat storage arcs.

In a similar manner as before, we arrive at the following (globally valid) inequality

$$\sum_{i \in V^h} \sum_{q=1}^{t_{max}-1} z_{iq}^W + \sum_{i \in V^h} \sum_{\substack{\{j \in (V^g \cup V^e):\\ (j,i) \in E\}}} \sum_{q=1}^{t_{max}} z_{jiq} \geq \left\lceil \sum_{i \in V^h} \frac{b_{it_{max}}}{\mathrm{M}_i^h} \right\rceil \tag{5.18}$$

Similar to how we derived Equation (5.17), we can extend this result to multiple time periods by taking heat storage into account. Let

$$\mathrm{M}_{it}^h = \begin{cases} \max(\max\{(\mu_{ji} \cdot u_{ji}) |\ (j,i) \in E : j \in V^g \cup V^e\}, \xi_h^2 \cdot \eta_h \cdot u^h), & \text{if } t > 1 \\ \max\{(\mu_{ji} \cdot u_{ji}) |\ (j,i) \in E : j \in V^g \cup V^e\}, & \text{else .} \end{cases}$$

We then arrive at the following VI, corresponding to the cuts in Figure 5.5.

$$\sum_{i \in V^h} \sum_{q=1}^{t-1} z_{iq}^W + \sum_{i \in V^h} \sum_{\tau=t}^{t_{max}} \sum_{\substack{\{j \in (V^g \cup V^e):\\ (j,i) \in E\}}} \sum_{q=1}^{\tau} z_{jiq} \geq \left\lceil \sum_{i \in V^h} \sum_{\tau=t}^{t_{max}} \frac{b_{i\tau}}{\mathrm{M}_{it}^h} \right\rceil, \quad \forall t \in T \tag{5.19}$$

5.3. Implementation

We investigate the influence of adding valid inequalites (5.17), (5.18) (5.19) to the MESDP. There are multiple ways in which these VI can be implemented. First, they can be added *a priori* to the model. Although this (possibly) improved polyhedral description of the feasible solution space results in tighter LP relaxations and therefore in better lower bounds from the start, directly adding all the VI is often not practical: it might become very difficult for the solver to generate feasible solutions, as the number of VI grows rapidly with the problem size. A second option is to add the VI dynamically when they are violated in an LP relaxation in a node in branch-and-bound tree. This can be implemented using callback functions. A third option is to add the VI a priori as *aggressive* lazy constraints. As mentioned before, regular lazy constraints are only pulled into the model when they are violated by a feasible solution. Aggressive lazy constraints are also pulled into the model when they are violated by a feasible solution in an *LP relaxation* . Only the VI that are violated by LP relaxations are pulled in, which might be more efficient as the solver does not have to deal with the entire set of VI in each iteration.

The results of these implementations can be found in Chapter 7

6

Decomposition methods

To solve large and complicated network design problems, the use of decomposition techniques can be convenient (Conejo et al., 2006). Through the application of decomposition techniques, certain types of problems can be solved in a decentralized or distributed fashion that can lead to a significant simplification of the problem. For a decomposition technique to be useful, the problem must have the appropriate structure. In practice, a distinction here is made between the following cases: the *complicating variable* and the *complicating constraint* structure. The MESDP can be viewed as a MILP with both a complicating variable and complicating constraint structure. In Section 6.1 and Section 6.2, we exploit the complicating variable and constraint structure of the MESDP, respectively.

6.1. Benders decomposition

Benders decomposition (BD) is based on the notion of *complicating variables*. The amount of literature on successful applications of this methodology to MI(L)Ps is extensive. The idea behind the BD method is to decompose a problem into two parts: the *master problem*, solving a relaxed version of the problem and obtaining values for a subset of the variables, and the *subproblem*, where values for the remaining variables are obtained while keeping the other ones fixed. Values of these remaining variables are then used to generate cuts for the master problem. Iteratively, the master- and the subproblem are solved, until no more cuts can be generated. Values found in the last iteration represent the optimal solution to the original problem. It is important to note that BD is only reasonable when both the master- and subproblem can be solved efficiently. This procedure is often applied to fixed charge network design problems, as the variables representing the opening of the links and the variables representing the flow of commodities form a natural decomposition scheme of a master- and subproblem, respectively (Costa, 2005).

6.1.1. Classical Benders decomposition

Since we are dealing with a MILP in this research, we present the theory of Benders decomposition for MILPs of the required structure. For a more general description of the BD method, we refer to Geoffrion (1972).

Consider the general formulation represented in (6.1), corresponding to most fixed-charge network design models in literature.

Minimize

$$c^T x + f^T y \tag{6.1}$$

Subject to

$$Ax + By \geq b$$
$$x \geq 0, y \in Y$$

Here, Y is a restricted set of potential solutions, often stating that y should be integer-valued. The variables $y \in Y$ are considered *complicating variables*: a known solution for them would reduce the total problem to an easier optimization problem that may be solved efficiently. For our explanation of the BD algorithm, we consider the case in which the problem reduces to an LP.

We can reformulate (6.1) as follows:

$$\min_{y \in Y}\{f^T y + \min_{x \geq 0}\{c^T x : Ax \geq b - By\}\}. \tag{6.2}$$

The inner minimization problem in (6.2) is a continuous linear program, which is known as the *Benders (primal) subproblem*. Associating dual variables u to constraints $Ax \geq b - By$, we can write the dual version of this problem as

$$\max_{u \geq 0}\{u^T(b - B\bar{y}) : A^T u \leq c\}, \tag{6.3}$$

for a fixed $\bar{y}$.

Problem (6.3) is known as the *Benders dual subproblem*.

Let $\bar{y}$ denote a fixed solution to the master problem. By strong duality, $u^{*T}(b - B\bar{y}) = c^T x^*$, i.e., the objective value of the optimal solution to the dual subproblem is the same as the objective value of the optimal solution to the primal subproblem for $\bar{y}$ (Geoffrion, 1972). Therefore, the primal and dual formulations can be interchanged and we can write (6.2) as

$$\min_{y \in Y}\{f^T y + \max_{u \geq 0}\{u^T(b - By) : A^T u \leq c\}\}. \tag{6.4}$$

If the primal subproblem is infeasible for a given $\bar{y}$, then we know that the dual subproblem

is either unbounded or infeasible. Since the constraints of the dual subproblem do not depend upon $\bar{y}$, we know that if the dual subproblem is infeasible for a single $\bar{y} \in Y$, that it is infeasible for all $y \in Y$. This in turn would imply that the primal subproblem is infeasible or unbounded for all $y \in Y$, and thus, that the original problem (6.1) is either infeasible or unbounded. In other words: if the original problem has a feasible solution, then we know that the dual subproblem is either bounded or unbounded, i.e., has a non-empty feasible space. Let $F = \{u \mid u \geq 0,\ A^T u \leq c\}$ denote this feasible space. We know that the non-empty polyhedron F is composed of extreme points u^p (for $p = 1, \ldots, P$) and extreme rays r^q (for $q = 1, \ldots Q$) (Bertsimas and Tsitsiklis, 1998). If the dual subproblem is bounded, then a solution u represents one of the extreme points u^p. If the dual subproblem is unbounded, then there is a direction r^q for which $r^q(b - B\bar{y}) > 0$ (Costa, 2005). This situation corresponds to an unfeasible primal problem and must be avoided. Therefore, the values of $\bar{y}$ corresponding to an unbounded dual subprogram must be eliminated. By explicitly considering the restrictions

$$r^q(b - B\bar{y}) \leq 0, \qquad q = 1, \ldots, Q, \tag{6.5}$$

we can prevent this from happening.

In formulation (6.4), the maximum value of the inner problem is the value of one of the extreme points of F. We can rewrite this problem as

$$\begin{aligned} &\min_{y \in Y}\{f^T y + \max_{u \geq 0}\{u^T(b - By) : A^T u \leq c\}\} \\ &\quad \text{s.t. } r^q(b - By) \leq 0, \qquad q \in 1, \ldots, Q, \end{aligned} \tag{6.6}$$

or, with some continuous variable z:

Minimize

$$f^T y + z \tag{6.7a}$$

subject to

$$z \geq u^p(b - By), \qquad p \in 1, \ldots, P, \tag{6.7b}$$

$$r^q(b - By) \leq 0, \qquad q \in 1, \ldots, Q, \tag{6.7c}$$

$$y \in Y, z \geq 0. \tag{6.7d}$$

Formulation (6.7) is known as the Benders reformulation. In this formulation, constraints (6.7b) provide an underestimate of the objective value of the Benders primal subproblem and con-

straints (6.7c) eliminate values of y that are infeasible for the original problem. The auxiliary variable z serves as an underestimator of the optimal objective value of the subproblem.

A limitation in solving (6.7) directly is the often extremely large set of constraints (6.7b)-(6.7c). To overcome this limitation, these constraints are generated iteratively. Initially, only constraints (6.7d) are considered, i.e., the first *Benders master problem*,

Minimize

$$f^T y + z \tag{6.8a}$$

subject to

$$y \in Y, z \geq 0, \tag{6.8b}$$

is solved. Problem (6.8) is a relaxed version of (6.7), i.e., for a solution $(\bar{y}, \bar{z})$ of (6.8), we know that $f^T \bar{y} + \bar{z}$ provides a lower bound to the obective of original problem (6.7).

In the first iteration, the solution of the first master problem $\bar{y}$ is inserted in dual subproblem (6.3). Solving the dual subproblem with either yields an unbounded solution, in which case a constraint of type (6.7c) is inserted in the master problem (known as a *Benders feasibility cut*); or finds a solution that is an extreme point, in which case the solution of the primal subproblem and the master problem form a complete solution and an upper bound to the original problem, while the solution of the dual subproblem is used to generate a constraint of type (6.7b) (known as a *Benders optimality cut*). The master- and subproblems are solved iteratively, until the upper- and lower bounds are sufficiently close.

Summarizing the above, the classical algorithm for implementing Benders decomposition is as follows:

Algorithm 1: Benders algorithm

Start with the original problem $\min\{c^T x + f^T y : Ax + By \geq b, x \geq 0, y \in Y\}$
Solve the initial master problem: $\min\{f^T y + z : y \in Y, z \geq 0\}$
Let $(\bar{y}, \bar{z})$ denote the optimal solution to the first master problem
Lower bound $(LB) := -\infty$
Upper bound $(UB) := \infty$
while $UB - LB > \epsilon$ **do**
    solve dual subproblem $\max_{u \geq 0}\{u^T(b - B\bar{y}) : A^T u \leq c\}$
    if *subproblem is unbounded* **then**
        Get unbounded ray $\bar{u}$
        Add cut $\bar{u}^T(b - By) \leq 0$ to the master problem
    else
        Get extreme point $\bar{u}$
        Add cut $z \geq \bar{u}^T(b - By)$ to the master problem
        $UB := \min\{UB, f^T\bar{y} + (b - B\bar{y})^T\bar{u})\}$
    end
    Solve the updated master problem $\min\{f^T y + z : cuts, y \in Y, z \geq 0\}$
    Let $(\bar{y}, \bar{z})$ denote the optimal solution to the updated master problem
    $LB := \bar{z} + f^T\bar{y}$
end

6.1.2. Improving the Benders decomposition method

The classical BD method as given in Algorithm 1 often does not perform well without the inclusion of various (problem-specific) acceleration techniques (Rahmaniani et al., 2017). The main drawbacks of the classical algorithm include: time consuming iterations; poor cuts; ineffective initial iterations and slow convergence at the end of the algorithm. A large and growing body of literature on BD is dedicated to exploring ways in which the convergence of the classical BD algorithm could be improved. In literature reviews by Costa (2005) and Rahmaniani et al. (2017), various computational enhancements to different components of the BD method are discussed. Listing all of these is beyond the scope of this research. Therefore, we only discuss extensions to the BD method that are relevant for applying Benders decomposition to the MESDP.

Disaggregation of the Benders sub problem In some cases, part of the constraint matrix A in the Benders sub problem is block-diagonal, implying that this problem can be partitioned into a set of independent optimization problems, one corresponding to each block. This frequently occurs in practice, when the subproblems contain a number of independent decisions that can be solved individually. In each iteration, the disaggregated cuts formed by the dual variables of the independent subproblems are added to the master problem simultaneously. This results in numerous benefits: solving a set of subproblems is often faster than solving one aggregated subproblem, and the cuts generated from the disaggregated problem are tighter than the ones from the original problem (Pearce and Forbes, 2018).

Benders within a branch and cut framework Solving the master problem (6.7) to optimality in each iteration can be computationally costly (Fragkos et al., 2017). Taking advantage of callback capabilities MIP solvers, it is possible to solve the master problem only once, generate cuts every time a feasible solution is found, add these as lazy constraints to the current set of cuts and return control to the solver. Note that in this method, it is necessary to add the Benders cuts as lazy constraints, as it is not possible to add user cuts that cut off feasible solutions from callback functions, as explained in Chapter 5.

Initial cuts The master problem is a relaxation of the original model. A master problem with fewer constraints is easier to solve, but it might also result in relaxed solutions that are far from the optimum in the first iterations of the BD algorithm, and therefore possibly in an algorithm that requires much more iterations. By providing a set of initial Benders cuts as well as additional valid inequalities to the initial problem, one might find the optimal solution in less iterations.

Two phase Benders To come up with such a collection of initial cuts for the first master problem, the Benders *two phase* algorithm can be useful (McDaniel and Devine, 1977). In the first phase, it applies the BD algorithm to the LP relaxation of the original problem. Note that these cuts are also valid for the original problem. When the BD algorithm for the LP relaxation has terminated, one can proceed to the second phase, in which the original problem is solved. In the intial master problem, the cuts generated in the first phase are added. Often, the LP relaxation is easier to solve using BD, and in addition generates a "good" set of cuts for the initial master problem, making two phase Benders a promising approach for MILP problems.

Selecting a branching direction When BD is implemented within a branch and cut framework using callback functions, it is possible to force the solver to always explore a particular branch first; either up or down. In some problems, always selecting a certain branching direction mield yield some benefit (Pearce and Hons, 2019).

6.1.3. Benders decomposition for the MESDP

We now introduce a Benders decomposition algorithm for the MESDP. First, we first introduce the (classical) BD of the MESDP. Second, we propose ways to enhance its performance.

Classical Benders decomposition

For the MESDP, the integer design variables $\{(z_{ijt}, z^s_{it}, z^W_{it}) : (i,j) \in E, i \in V, t \in T\}$ (throughout this section referred to as z for notation purposes) and binary variables $\{\gamma_{ijt} : (i,j) \in E, t \in T\}$ are complicating variables. Let the initial master problem for the MESDP therefore be given by

Minimize

$$\sum_{t\in T}\Big(\sum_{(i,j)\in E} z_{ijt}\cdot c_{ijt} + \sum_{k\in K}\sum_{i\in\bigcup_{k\in K}V^k}\Big(\sum_{s} z^s_{it}\cdot c^s_t + z^W_{it}\cdot c^W_t\Big)\Big) \tag{6.9a}$$

subject to

$$z_{ijt} = z_{jit}, \qquad \forall (i,j)\in E^N,\ \forall t\in T \tag{6.9b}$$

$$z^s_{it} = 0, \quad \text{if } i\notin V^e \quad \forall i\in V, \forall t\in T \tag{6.9c}$$

$$z_{ijt} = z_{iht}, \quad \forall i\in V^g, \forall j\in V^e, h\in V^h, \forall t\in T \tag{6.9d}$$

$$z_{ijt},\ z^s_{it},\ z^W_{it}\in\mathbb{Z}^+, \gamma_{ijt}\in\{0,1\}, \tag{6.9e}$$

A solution $(\bar{z},\bar{\gamma})$ denotes a feasible solution for the MESDP if and only if there exist energy flow-, supply- and storage variables $x_{ijt}, S_{it}, W^{IN}_{it}$ and W^{OUT}_{it}, $(i,j)\in E, i\in V, t\in T$ satisfying the inequalities in the original model (3.1) given by (3.1a), (3.1b), (3.1d), (3.1f), (3.1g), (3.1k), (3.1l), (3.1i) for $(\bar{z},\bar{\gamma})$. Checking whether $(\bar{z},\bar{\gamma})$ is feasible, corresponds to finding a feasible flow in the time expanded graph (where the capacities of all assets and the direction of flow on the network edges is fixed). Note that this problem can be reduced to a network flow problem and solved with any max-flow algorithm (Ahuja et al., 1993).

For a fixed solution $(\bar{z},\bar{\gamma})$ the Benders sub-problem is

Minimize

$$0 \tag{6.10a}$$

subject to

$$\sum_{\{j:(j,i)\in E\}} \mu_{ji} \cdot x_{jit} - \sum_{\{j:(i,j)\in E\}} x_{ijt} + S_{it} - W_{it}^{IN} + W_{it}^{OUT} = b_{it}, \quad \forall i \in V, \forall t \in T \tag{6.10b}$$

$$S_{it} \leq \sum_{s} \left(u_t^s \cdot \sum_{q=1}^{t} \bar{z}_{iq}^s \right) + g_{it}, \quad \forall i \in V,\ \forall t \in T \tag{6.10c}$$

$$x_{ijt} \leq u_{ij} \cdot \sum_{q=1}^{t} \bar{z}_{ijq}, \quad \forall (i,j) \in E,\ \forall t \in T \tag{6.10d}$$

$$x_{ijt} \leq \mathcal{M} \bar{\gamma}_{ijt}, \quad \forall (i,j) \in E^N,\ \forall t \in T \tag{6.10e}$$

$$x_{ijt} \leq \mathcal{M}(1 - \bar{\gamma}_{jit}), \quad \forall (i,j) \in E^N,\ \forall t \in T \tag{6.10f}$$

$$W_{it}^{IN} \leq u^k \cdot \sum_{q=1}^{t} \bar{z}_{iq}^W - \sum_{q=1}^{t-1} \left(\xi_k \cdot W_{iq}^{IN} \cdot \eta_k^{t-q} - W_{iq}^{OUT} \right), \quad \forall i \in V^k,\ \forall l,k \in K,\ \forall t \in T \tag{6.10g}$$

$$W_{it}^{OUT} \leq \xi_k \cdot \sum_{q=1}^{t-1} \left(\xi_k \cdot W_{iq}^{IN} \cdot \eta_k^{t-q} - W_{iq}^{OUT} \right), \quad \forall i \in V^k, \forall k \in K, \forall t \in T \tag{6.10h}$$

$$x_{ijt} = x_{iht}, \quad \forall (i,j),(i,h) \in E^C : i \in V^g, \forall t \in T \tag{6.10i}$$

$$x_{ijt},\ S_{it},\ W_{it}^{IN},\ W_{it}^{OUT} \in \mathbb{R}^+, \qquad \forall (i,j) \in E, \forall i \in V, \forall t \in T, \tag{6.10j}$$

Note that this is a feasibility problem with a “dummy” objective. If, for an optimal solution $(\bar{z}, \bar{\gamma})$ of the master problem, we have a feasible subproblem, then an optimal solution for the original problem is found, as we are minimizing an objective function with (integer) design variables z only.

Let $\alpha_{it}, \beta_{it}, \pi_{ijt}, \omega_{ijt}^1, \omega_{ijt}^2, \sigma_{it}^{IN}, \sigma_{it}^{OUT}$ and τ_{ijt} denote the dual variables for constraints (6.10b), (6.10c), (6.10d), (6.10e), (6.10f), (6.10g), (6.10h) and (6.10i), respectively.

The Benders dual sub-problem for the MESDP is given by:

Maximize

$$\begin{aligned} &\sum_{i\in V}\sum_{t\in T} b_{it} \cdot \alpha_{it} + \sum_{i\in V}\sum_{t\in T}\left(\sum_{s} \left(u_t^s \cdot \sum_{q=1}^{t} \bar{z}_{iq}^s \right) + g_{it} \right) \cdot \beta_{it} + \\ &\sum_{(i,j)\in E}\sum_{t\in T}\left(u_{ij} \cdot \sum_{q=1}^{t} \bar{z}_{ijq} \right) \cdot \pi_{ijt} + \sum_{k\in K}\sum_{i\in V^k}\sum_{t\in T} \sigma_{it}^{IN} \cdot \left(u^k \sum_{q=1}^{t} \bar{z}_{iq}^k \right) + \\ &\qquad \sum_{(i,j)\in E^N}\sum_{t\in T} \mathcal{M} \cdot \left(\bar{\gamma}_{ijt} \cdot \omega_{ijt}^1 + (1 - \bar{\gamma}_{ijt}) \cdot \omega_{ijt}^2 \right) \end{aligned} \tag{6.11a}$$

subject to

$$\mu_{ij}\alpha_{jt} - \alpha_{it} + \pi_{ijt} + \omega^1_{ijt} + \omega^2_{jit} \leq 0, \quad \forall (i,j) \in E^N, \forall t \in T \tag{6.11b}$$

$$\mu_{ij}\alpha_{jt} - \alpha_{it} + \pi_{ijt} + \tau_{ijt} - \tau_{iht} \leq 0, \quad \forall (i,j),(i,h) \in E^C : i \in V^g, \forall t \in T \tag{6.11c}$$

$$\mu_{ij}\alpha_{jt} - \alpha_{it} + \pi_{ijt} \leq 0, \quad \forall (i,j),(i,h) \in E^C : i \notin V^g, \forall t \in T \tag{6.11d}$$

$$\alpha_{it} + \beta_{it} \leq 0, \qquad \forall i \in V, \forall t \in T \tag{6.11e}$$

$$-\alpha_{it} + \sigma^{IN}_{it} + \xi_k \cdot \sum_{q=t+1}^{t_{max}} \eta_k^{q-t} \cdot \left(\sigma^{IN}_{iq} - \xi_k \cdot \sigma^{OUT}_{iq}\right) \leq 0, \qquad \forall k \in K, \forall i \in V^k, \forall t \in T \tag{6.11f}$$

$$\alpha_{it} + \sigma^{OUT}_{it} + \sum_{q=t+1}^{t_{max}} \left(\xi_k \cdot \sigma^{OUT}_{iq} - \sigma^{IN}_{iq}\right) \leq 0, \qquad \forall k \in K, \forall i \in V^k, \forall t \in T \tag{6.11g}$$

$$\beta_{it}, \pi_{ijt}, \omega^1_{ijt}, \omega^2_{ijt}, \sigma^{IN}_{it}, \sigma^{OUT}_{it} \in \mathbb{R}^-, \qquad \forall (i,j) \in E, \forall i \in V, \forall t \in T \tag{6.11h}$$

$$\alpha_{it}, \tau_{ijt} \in \mathbb{R}, \qquad \forall (i,j) \in E, \forall i \in V, \forall t \in T \tag{6.11i}$$

Corresponding to Algorithm 1, we iteratively solve problems (6.9) and (6.11). When problem (6.11) is unbounded, we retrieve an unbounded ray ($\alpha_{it}, \beta_{it}, \pi_{ijt}, \omega^1_{ijt}, \omega^2_{ijt}, \sigma^{IN}_{it}, \sigma^{OUT}_{it}, \tau_{ijt}$) and add the following (feasibility) cut to the master problem (6.9):

$$\begin{aligned} &\sum_{i \in V}\sum_{t \in T} b_{it} \cdot \bar{\alpha}_{it} + \sum_{i \in V}\sum_{t \in T}\left(\sum_s \left(u^s_t \cdot \sum_{q=1}^{t} z^s_{iq}\right) + g_{it}\right) \cdot \bar{\beta}_{it} + \\ &\sum_{(i,j) \in E}\sum_{t \in T}\left(u_{ij} \cdot \sum_{q=1}^{t} z_{ijq}\right) \cdot \bar{\pi}_{ijt} + \sum_{k \in K}\sum_{i \in V^k}\sum_{t \in T} \bar{\sigma}^{IN}_{it} \cdot \left(u^k \sum_{q=1}^{t} z^k_{iq}\right) + \\ &\sum_{(i,j) \in E^N}\sum_{t \in T} \mathcal{M} \cdot \left(\gamma_{ijt} \cdot \bar{\omega}^1_{ijt} + (1 - \gamma_{ijt}) \cdot \bar{\omega}^2_{ijt})\right) \leq 0. \end{aligned} \tag{6.12}$$

When problem (6.11) is bounded for a given master solution $\bar{y}$, we know that a primal feasible solution must exist for $\bar{y}$ and that this primal feasible solution has to be optimal for the original problem.

Implementing the classical version of BD as given in Algorithm 1 turns out to be highly inefficient for solving even small cases of the MESDP. However, as mentioned in Section 6.1.2, this is not surprising as the classical BD algorithm often performs poor. Therefore, we include various computational enhancements in our BD algorithm for the MESDP. We first introduce the computational enhancements that were considered, and subsequently formally introduce a BD algorithm for the MESDP.

Computational enhancements for the Benders decomposition of the MESDP

Disaggregation Multi-period network designs often have a structure that lends itself nicely for a disaggregated decomposition. For BD specifically, many examples can be thought of in which the "flow" subproblems in different time periods do not depend on each other and can all be solved individually. For the MESDP, although this is a multi-period network problem, this is not exactly the case, as flow can travel between time periods through the use of storage. It is however possible to decompose the subproblems into a series of $|T|$ subproblems, since the flow in each time period only depends on the *previous* time periods. Using this fact, we can decompose polyhedron (6.10b) - (6.13i) of the Benders subproblem (6.11) into $|T|$ "sub"-polyhedra as follows:

$$\sum_{\{j:(j,i)\in E\}} \mu_{ji} \cdot x_{jit} - \sum_{\{j:(i,j)\in E\}} x_{ijt} + S_{it} - W_{it}^{IN} + W_{it}^{OUT} = b_{it}, \quad \forall i \in V, \forall t \in T^{\tau} \tag{6.13a}$$

$$S_{it} \leq \sum_{s} \left(u_t^s \cdot \sum_{q=1}^{t} \bar{z}_{iq}^s \right) + g_{it}, \quad \forall i \in V,\ \forall t \in T^{\tau} \tag{6.13b}$$

$$x_{ijt} \leq u_{ij} \cdot \sum_{q=1}^{t} \bar{z}_{ijq}, \quad \forall (i,j) \in E,\ \forall t \in T^{\tau} \tag{6.13c}$$

$$x_{ijt} \leq \mathcal{M} \bar{\gamma}_{ijt}, \quad \forall (i,j) \in E^N,\ \forall t \in T^{\tau} \tag{6.13d}$$

$$x_{ijt} \leq \mathcal{M}(1 - \bar{\gamma}_{jit}), \quad \forall (i,j) \in E^N,\ \forall t \in T^{\tau} \tag{6.13e}$$

$$W_{it}^{IN} \leq u^k \cdot \sum_{q=1}^{t} \bar{z}_{iq}^W - \sum_{q=1}^{t-1} \left(\xi_k \cdot W_{iq}^{IN} \cdot \eta_k^{t-q} - W_{iq}^{OUT} \right), \quad \forall i \in V^k,\ \forall l,k \in K,\ \forall t \in T^{\tau} \tag{6.13f}$$

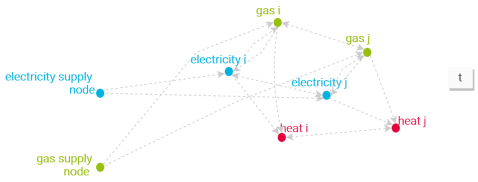
$$W_{it}^{OUT} \leq \xi_k \cdot \sum_{q=1}^{t-1} \left(\xi_k \cdot W_{iq}^{IN} \cdot \eta_k^{t-q} - W_{iq}^{OUT} \right), \quad \forall i \in V^k, \forall k \in K, \forall t \in T^{\tau} \tag{6.13g}$$

$$x_{ijt} = x_{iht}, \quad \forall (i,j), (i,h) \in E^C : i \in V^g, \forall t \in T^{\tau} \tag{6.13h}$$

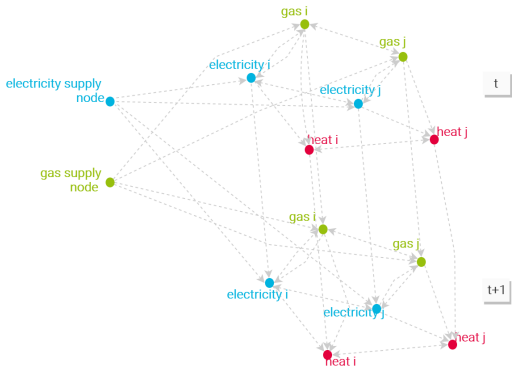
$$x_{ijt},\ S_{it},\ W_{it}^{IN},\ W_{it}^{OUT} \in \mathbb{R}^+, \qquad \forall (i,j) \in E, \forall i \in V, \forall t \in T^{\tau}, \tag{6.13i}$$

where for each $\tau \in T$ we have $T^{\tau} = \{t : t \leq \tau, t \in T\}$, i.e., T^{τ} consists of τ and all the time periods before τ. For the corresponding disaggregated dual sub problems we replace T in (6.11) with T^{τ} .

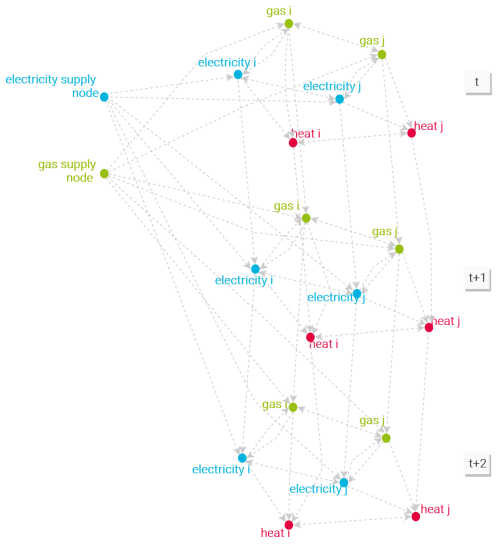
As mentioned before, checking whether a Benders subproblem is feasible for the given capacities $(\bar{z}, \bar{\gamma})$ corresponds to checking whether a feasible flow exists in the time-expanded graph G^T. When we decompose the subproblem into $|T|$ smaller subproblems, we should therefore check whether a feasible flow exists in each time-expanded graph $G^{T^{\tau}}$ (Figure 6.1).



(a) The time expanded graph for T^t



(b) The time expanded graph for T^{t+1}



(c) The time expanded graph for T^{t+2}

Figure 6.1: Decomposition of the Benders subproblem corresponds to solving a flow feasibily problem for each time-expanded graph.

In each iteration of the BD, all $|T|$ subproblems are considered and violated cuts are generated from each of them. Computational experiments showed that the disaggregated formulation is strongly preferred of the aggregated version for the MESDP, as the disaggregated cuts are significantly tighter. For the remainder of this chapter, we will therefore consider the disaggregated version.

Benders within a branch and cut framework Even with the disaggregated BD, computational experiments showed that the classical BD algorithm converged rather slowly. The callback version outperformed the classical algorithm and is therefore preferred for the MESDP.

Initial cuts To start with a good set of initial cuts for the master problem, we propose to do the following:

- include a set of valid inequalities, consisting of:
  - the valid inequalities (5.10), (5.17), (5.19), from Chapter 5;
  - an additional set of valid inequalities;
- run the Benders two phase method, i.e., solve the LP relaxation of the MESDP using BD first to generate a set of initial Benders cuts.

The additional set of VI consists of constraints that are redundant for the original MESDP, but tighten the initial feasible space of the Benders master problems through additional constraints on the z variables. In the regular formulation of the MESDP, the z variables are constrained by the values of the flow variables. As they are not present in the Benders master

problem, it is useful to pose some extra constraints on z, to decrease the solution space for the integer variables. To this end, we propose to add the following inequalities to the initial Benders master problem of the MESDP:

$$\sum_{i\in V}\sum_{\tau=t}^{t_{max}}\sum_{s} u^s \cdot \sum_{q=1}^{\tau} z_{iq}^s + \sum_{k\in K}\sum_{i\in V^k} \xi_k^2 \cdot \eta_k \cdot u^k \cdot \sum_{q=1}^{t-1} z_{iq}^W \geq \sum_{i\in V}\sum_{\tau=t}^{t_{max}} \left(\frac{b_{i\tau}}{\mathrm{M}_i} - g_{i\tau}\right), \qquad \forall t \in T \quad (6.14)$$

and

$$\sum_{\{j:(j,i)\in E\}} \mu_{ji} \cdot u_{jit} \cdot \sum_{q=1}^{t} z_{jiq} + \sum_{s} u^s \cdot \sum_{q=1}^{t} z_{iq}^s + \xi_k^2 \cdot \eta_k \cdot u^k \cdot \sum_{q=1}^{t-1} z_{iq}^W + g_{it} \geq b_{it}, \quad \forall i \in \bigcup_{k\in K} V^k, \ \forall t \in T. \quad (6.15)$$

Note that these are valid (and redundant) for the original formulation of the MESDP.

A BD algorithm for the MESDP

We propose to iteratively solve the master- and subproblems with Algorithm 2:

Algorithm 2: Benders algorithm for the MESDP

PHASE 1
Solve the LP relaxation of initial master problem (6.9)
Let $(\bar{z}, \bar{\gamma})$ denote the optimal solution
for *a certain number of iterations* **do**
  for $\tau \in T$ **do**
    Check whether subproblem τ (6.13) is feasible for $(\bar{z}, \bar{\gamma})$
    if *Subproblem τ is infeasible* **then**
      Add cut (6.12) to the master problem
    end
  end
  Solve the updated relaxed master problem { (6.9) | $cuts$}
  Let $(\bar{z}, \bar{\gamma})$ denote the optimal solution to the master problem
  if *objective has not improved for a number of iterations or max iterations reached* **then**
    exit algorithm
  else
end
PHASE 2
Let $cuts$ denote the set of cuts generated in PHASE 1.
Optimize the (integer) initial master problem { (6.9) | $cuts$}
if *found new incumbent solution* **then**
  Let $(\bar{z}, \bar{\gamma})$ denote the incumbent solution
  for $\tau \in T$ **do**
    Check whether subproblem τ (6.13) is feasible for $(\bar{z}, \bar{\gamma})$
    if *Subproblem τ is infeasible* **then**
      Add lazy constraint (6.12) to the master problem
    end
  end
else
end

6.2. Lagrangean Relaxation

Lagrangean relaxation (LR) is based on the notion of *complicating constraints*. In network optimization problems, LR has many applications. The class of networks with side constraints (for instance certain relationships between several of the arcs in a network) is a good candidate for LR. Also for network design problems with integer values, LR is a commonly applied solution approach. The reason for this is that LR is attractive for solving MILPs where the gap between the LP-relaxation and the integer programming value is large, and this is often the case for the MILP formulation of network design problems (Magnanti et al., 1991). In this sec-

tion, we first introduce some general theory regarding LR. Second, we propose a way in which LR can be used to find good lower bounds for the MESDP.

6.2.1. Theory

To illustrate the LR method in a general way, suppose that we have the following optimization model, where x is a vector of decision variables.

Minimize

$$z^* = c^T x \tag{6.16a}$$

subject to

$$Ax = b \tag{6.16b}$$

$$x \in X \tag{6.16c}$$

In model (6.16), there is a linear objective function and a set of explicit linear constraints (6.16b). In addition, decision variables x lie in a given constraint set X (6.16c).

The LR method relaxes certain constraints by bringing them into the objective function with associated Lagrange multipliers μ. We then find the following problem:

Minimize

$$c^T x + \mu(Ax - b) \tag{6.17a}$$

subject to

$$x \in X \tag{6.17b}$$

The resulting problem (6.17) is referred to as a *Lagrangean relaxation* or a *Lagrangean subproblem* of the original problem. We refer to

$$L(\mu) = \min\{c^T x + \mu(Ax - b) : x \in X\} \tag{6.18}$$

as the *Lagrangean function*. We can state the following about the Lagrangean function (Ahuja et al., 1993):

Lemma 2 (Lagrangean Bounding Principle)**.** *For any vector μ of the Lagrangean multipliers, the value $L(\mu)$ of the Lagrangean function is a lower bound on the optimal objective function value z^* of problem* (6.16)*.*

Proof. For the proof we refer to Ahuja et al. (1993). □

To find the *best lower bound* on the objective value of the original problem (6.16), it is necessary to solve the *Lagrangean multiplier problem*:

$$L^*(\mu) = \max_{\mu} L(\mu) \tag{6.19}$$

When relaxing inequality constraints instead of equality constraints ($Ax \leq b$ in constraint (6.16b)), the Lagrange multiplier problem becomes

$$L^* = \max_{\mu \geq 0} L(\mu). \tag{6.20}$$

The Lagrangean multiplier problem (6.20) is often solved through the use of the *subgradient method*, due to its convergence properties and because it is easily implemented and works for many types of problems (Bertsekas, 1998).

6.2.2. The subgradient method

The subgradient method is an adaptation of the gradient method in which gradients are replaced by subgradients (Fisher, 2004). Given an initial value μ^0 of a Lagrange multiplier, a sequence of $\{\mu^k\}$ is generated by

$$\mu^{k+1} = \mu^k + t^k(Ax^k - b),$$

where x^k is an optimal solution to $L(\mu^k)$ and t^k is a positive scalar step size. A theoretical result on the convergence of the subgradient method is that $L(\mu^k) \to L^*$ if $t^k \to 0$ and $\sum_{i=0}^{k} t^k \to \infty$. The step size that is most commonly applied in practice is

$$t^k = \frac{\beta^k(Z^* - L(\mu_k))}{\|g^k\|^2},$$

where β^k is a scalar satisfying $0 < \beta^k \leq 2$ and Z^* is an upper bound on L^*, which is often obtained by applying a heuristic method to the original problem (6.16), and $g^k = Ax^k - b$ (the "size" of the violation in the current solution). For more information on the subgradient method and the derivation of good step sizes, we refer to Held et al. (1974). Often, initially $\beta^0 = 2$, and gets halved whenever $L(\mu^k)$ fails to increase for a certain number of iterations. A large body of empirical findings justify this choice for the β^k , but it might not always be the best option (Fisher, 2004). When a feasible solution to the original problem is found in a Lagrangean

problem that is smaller than the current Z^*, then Z^* gets updated. The subgradient method is usually terminated after a certain number of iterations.

6.2.3. Lagrangean relaxation for the MESDP

For the MESDP, there are multiple candidate constraints to be relaxed in a LR. First, it is possible to apply a so-called *knapsack relaxation* (Gendron, 2011), by relaxing flow conservation constraints (3.1a) in the original formulation of the MESDP. Second, a *shortest path relaxation* can be applied, in which all constraints linking the flow-and design variables are relaxed, i.e., constraints (3.1b), (3.1d), (3.1f), (3.1g) and (3.1k). Both of these strategies might result in a tight lower bound obtained from the LR, but we choose to only investigate the latter here, as this strategy seems most promising based on research into similar problems.

A LR of the MESDP, with Lagrangean multipliers λ is therefore given by

Minimize

$$
\begin{aligned}
L(\lambda) = &\sum_{t\in T}\Big(\sum_{(i,j)\in E} z_{ijt}\cdot c_{ijt} + \sum_{i\in V}\Big(\sum_{s\in S} z^s_{it}\cdot c^s_t + \sum_{k\in K} z^W_{it}\cdot c^W_t\Big)\Big) + \\
&\sum_{i\in V}\sum_{t\in T}\lambda^1_{it}\cdot\Big(S_{it} - \sum_{s}\Big(u^s_t\cdot\sum_{q=1}^{t} z^s_{iq}\Big) + g_{it}\Big) + \sum_{(i,j)\in E}\sum_{t\in T}\lambda^2_{ijt}\cdot\Big(x_{ijt} - u_{ij}\cdot\sum_{q=1}^{t} z_{ijq}\Big) + \\
&\sum_{(i,j)\in E^N}\sum_{t\in T}\lambda^3_{ijt}\cdot\Big(x_{ijt} - \mathcal{M}\gamma_{ijt}\Big) + \sum_{(i,j)\in E^N}\sum_{t\in T}\lambda^4_{ijt}\cdot\Big(x_{ijt} - \mathcal{M}(1-\gamma_{jit})\Big) + \\
&\sum_{i\in\bigcup_{k\in K}V^k}\sum_{t\in T}\lambda^5_{it}\cdot\Big(W^{IN}_{it} - u^k\cdot\sum_{q=1}^{t} z^W_{iq} - \sum_{q=1}^{t-1}\Big(\xi_k\cdot W^{IN}_{iq}\cdot\eta_k^{t-q} - W^{OUT}_{iq}\Big)\Big)
\end{aligned}
\tag{6.21a}
$$

subject to

$$
\sum_{\{j:(j,i)\in E\}}\mu_{ji}\cdot x_{jit} - \sum_{\{j:(i,j)\in E\}} x_{ijt} + S_{it} - W^{IN}_{it} + W^{OUT}_{it} = b_{it}, \quad \forall i\in V, \forall t\in T \tag{6.21b}
$$

$$
z^s_{it} = 0, \quad \text{if } i\notin V^e \quad \forall i\in V, \forall t\in T \tag{6.21c}
$$

$$
z_{ijt} = z_{jit}, \qquad \forall (i,j)\in E^N,\ \forall t\in T \tag{6.21d}
$$

$$
z_{ijt} = z_{iht}, \quad \forall (i,j),(i,h)\in E^C : i\in V^g, \forall t\in T \tag{6.21e}
$$

$$
x_{ijt} = x_{iht}, \quad \forall (i,j),(i,h)\in E^C : i\in V^g, \forall t\in T \tag{6.21f}
$$

$$
W^{OUT}_{it} \le \xi_k\cdot\sum_{q=1}^{t-1}\Big(\xi_k\cdot W^{IN}_{iq}\cdot\eta_k^{t-q} - W^{OUT}_{iq}\Big), \quad \forall i\in\bigcup_{k\in K}V^k, \forall t\in T \tag{6.21g}
$$

$$
z_{ijt},\ z^s_{it},\ z^k_{it}\in\mathbb{Z}^+, \quad x_{ijt},\ S_{it},\ W^{IN}_{it},\ W^{OUT}_{it}\in\mathbb{R}^+, \quad \gamma_{ijt}\in\{0,1\}, \tag{6.21h}
$$

For given multipliers $\lambda^1, \lambda^2, \lambda^3, \lambda^4$, and λ^5, problem (6.21) is significantly easier to solve than the original formulation (3.2) of the MESDP. By penalizing continuous variables S, x, W^{IN} in the objective function (6.21a) and noting that these variables correspond to flow in the time-expanded graph G^T, we have formulated a variation of a minimum cost flow problem in problem (6.21).

The corresponding Lagrange multiplier problem is

$$\max_{\lambda \geq 0} L(\lambda). \tag{6.22}$$

The solution to this maximization problem provides the best choice for multipliers λ, i.e., those that result in the best lower bound for the MESDP. We propose a subgradient method to solve (6.22) for the MESDP.

When applying the subgradient method to problem (6.21), we alternately solve it for a set of fixed multipliers λ, and subsequently update the multipliers using the algorithmic procedures of the subgradient method.

Often in LR, the multipliers are all set equal to zero in the first iteration. For the MESDP, we choose to also apply this strategy.

We propose the following algorithm for solving the Lagrange multiplier problem of the MESDP.

Algorithm 3: A subgradient method for solving the Lagrange multiplier problem of the MESDP

Input: An upper bound Z^* on the MESDP (3.2).

Initialize $\lambda^0 = 0$, $\beta^0 := 2$ and $L_{best} := -\infty$

for $p := 0, 1, \cdots$ **do**

calculate f^p as in (6.23)

$t^p := \dfrac{\beta^p (Z^* - L(\lambda^p))}{\|f^p\|^2}$

$\lambda^{p+1} = \max\{\lambda^p + t^p \cdot f^p, 0\}$

if $\|\lambda^{p+1} - \lambda^p\| < \epsilon$ **then**

Stop

end

$L_{best} = \max(L_{best}, L(\lambda^p))$

if $Z^* < L(\lambda^p)$ *and* $L(\lambda^p)$ *is feasible* **then**

$Z^* := L(\lambda^p)$

end

if *no progress in more than P iterations* **then**

$\beta^{p+1} := \beta^p \cdot \frac{1}{2}$

else

$\beta^{p+1} := \beta^p$

end

$p := p + 1$

end

return L_{best}

Where, in each iteration p, we have

$$
\begin{aligned}
f^p := \sum_{i \in V} \sum_{t \in T} \left((S_{it})^p - \sum_{s} \left(u_t^s \cdot \sum_{q=1}^{t} (z_{iq}^s)^p \right) + g_{it} \right) + \sum_{(i,j) \in E} \sum_{t \in T} \left(x_{ijt}^p - u_{ij} \cdot \sum_{q=1}^{t} z_{ijq}^p \right) + \\
\sum_{(i,j) \in E^N} \sum_{t \in T} \left(x_{ijt}^p - \mathcal{M} \gamma_{ijt}^p \right) + \sum_{(i,j) \in E^N} \sum_{t \in T} \left(x_{ijt}^p - \mathcal{M}(1 - \gamma_{jit}^p) \right) + \\
\sum_{i \in \bigcup_{k \in K} V^k} \sum_{t \in T} \left((W_{it}^{IN})^p - u^k \cdot \sum_{q=1}^{t} (z_{iq}^W)^p - \sum_{q=1}^{t-1} \left(\xi_k \cdot (W_{iq}^{IN})^p \cdot \eta_k^{t-q} - (W_{iq}^{OUT})^p \right) \right)
\end{aligned}
\tag{6.23}
$$

The upper bound Z^* can be found using heuristic methods. In our algorithm, we find an upper bound Z^* by letting the solver make the few first iterations of the branch and bound procedure for the original problem and then use the found incumbent solution.

Lagrangean relaxation is often incorporated into a heuristic. In such a heuristic, the solution

to the LR problem is attempted to be transformed into a feasible solution for the original problem. In this way, LR can be used to generate upper bounds for the original problem, by post-processing a solution to the LR.

In this research, we only apply LR to find lower bounds for the MESDP. We compare the lower bounds computed through this method with the lower bounds computed by the LP relaxation in Chapter 7. Should, through this method, tighter lower bounds be generated (in an easy way), then incorporating LR into the branch and bound method of the original problem (instead of LP relaxations) is an interesting option for the MESDP.

7

Computational results

In this chapter, we provide results of experiments with the different solution methods. First, we introduce the case study for which we conduct the experiments. Second, we illustrate how experiments with smaller instances of a base case provide us with insights in potential bottlenecks for the solver. Based on these, we generate a set of instances. Third, we provide results of the experiments with the solver's branch and cut method, both with parameter tuning as introduced in Chapter 4 and the addition of valid inequalities as introduced in Chapter 5. Lastly, we provide results of the experiments with the two decomposition methods, as introduced in Chapter 6, and compare these to the default branch and bound solver.

For all of the following experiments, we have coded the algorithms in Python 3.7.6. and implemented the model in Pyomo 5.7.0 , a Pythonic framework for formulating optimization models. Problem instances are solved with the commercial MIP solver Gurobi 8.1.1. The experiments are run on a laptop with Intel® Core™ i7-8650U CPU @ 1.90 GHz Processor with 16,0 GB RAM memory.

7.1. Casy study

Data from the city of Eindhoven is used throughout this research for the time period 2018-2050. With time steps of two years, a demand profile is generated for 110 locations in the city, corresponding to the 110 gas access points in Eindhoven. In each year and at each location, the demand has to be fulfilled for electricity, gas and heat. Based on the ambitious climate targets of the city of Eindhoven, future energy scenarios have been created. Costs and capacities of the different assets are generated based on different national and international databases (Van Beuzekom et al., 2020). By clustering these 110 locations, we create smaller instances of 28 and 7 locations. Proportional to these smaller instance sizes, we adjust the data of the demand profiles, supply, and asset costs and capacities. In these experiments, we focus mainly on solving problem instances of the MESDP with 7 and 28 locations. These locations roughly correspond to the city's districts and neighborhoods, respectively. In addition, for each solution method, we conducted an experiment with a single problem instance of the 110 node case.

The base case Data input for the base case is provided by Van Beuzekom (2020) and consists

of costs, capacities and technological efficiencies of the different assets, the demand of each location for each energy carrier in each time period, gas supply in each time period, a discount- and technological development rate, the conversion rates, loss factors of storage and losses over distance. Three sizes of the same base case are considered: the 7-, 28- and 110 node case, with accordingly scaled input data.

A set of problem instances As mentioned in Section 4.2, in order to assess the quality of the solution methods, a set of problem instances with variations on the base case should be generated. We generate this set of instances based on empirical results (Section 7.2).

In addition to these "empirically" generated instances, we generate an instance that takes into account the current gas- and electricity network. For this, we have conducted a detailed analysis of the energy infrastructure in the city of Eindhoven, using open data from Enexis (2019), GeoPandas version 0.7.0, the Python package for working with geospatial data, and QGIS Desktop version 3.10.2, an open source geographic information system. The findings are then translated to a starting network for the 7- and 28-location case, respectively. This represents a "free" network at $t = -1$. By including this network at $t = -1$, we do not exclude the model's possibility to increase the capacity of the starting network at $t = 0$ (and in the time periods that follow) [1]. As an example for the electricity network: Figure 7.1 is a graphical representation of the (medium-pressure) electricity network in Eindhoven, together with the coordinates of the 28 nodes. By examining this network in detail, we can decide which electricity edges in graph G^T should be part of a starting solution for the 28-node case.

7.2. Experiments with the Gurobi solver

This section provides the results of experiments with Gurobi's branch and cut algorithm. We perform the empirical study as introduced in Chapter 4, and subsequently investigate the effect of adding the valid inequalities. First, we examine the solver's output for a few small instances of the base case, i.e., containing less locations and/or time periods. By examining the output of these experiments, we tend to find out whether the solver has problems, and if so identify their cause(s). Based on these findings, we generate a set of problem instances with a variety of (pre-estimated) "easy" and "difficult" properties. With this set of instances, we run the parameter tuning experiments as introduced in Chapter 7.2.2 and the experiments with VI as introduced in Chapter 5.

Experiments with smaller instances We reduce the size of the regular instance by decreasing the number of locations and the number of time periods respectively. Without changing any other parameters, we have implemented instances of the base case with 7 locations while varying the number of time steps from 1 to 17, and using 17 time steps while varying the number of locations from 1 to 7. In these experiments, the measure of efficiency that we used was the time to solve to optimality. For cases with less time periods, we noticed the follow-

[1] It should be noted that, when modeled accurately, the network in the first time step is sufficient to meet all demand in the first time period and that the model should only build new assets if that is efficient for future time periods - we can (hopefully) assume that the current network in the city of Eindhoven should suffice to fulfill the demand in 2018, the first time period.

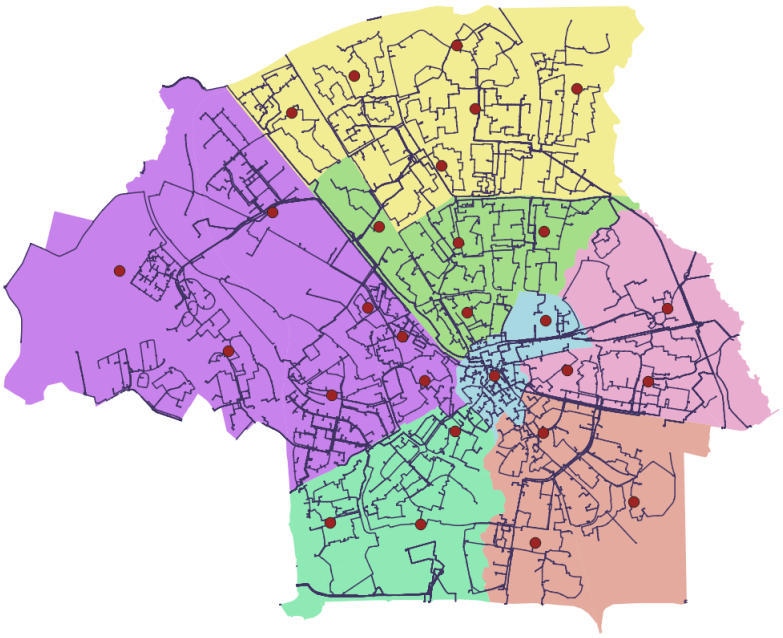


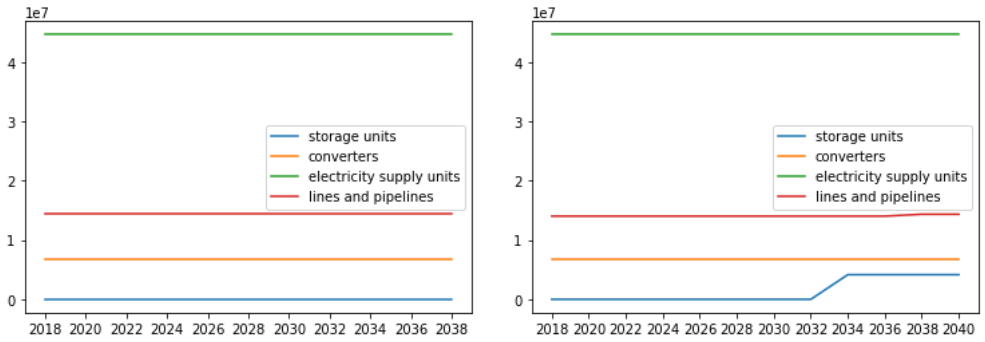
Figure 7.1: The current electricity network in the city of Eindhoven and the locations of the 28 nodes. By translating this network, as well as the ga network and the location of some existing storage-, supply- and conversion assets to a starting network for the MESDP, we can consider a Brownfield case.

ing: when the number of time steps is less than 12, the algorithm finds an optimal solution in less than 300 seconds. When the number of time steps is 12 or higher, the model cannot solve optimally within a reasonable amount of time anymore (even after letting the solver run for multiple days!). We can point out why this is the case by inspecting the solutions and the input data: after 11 time steps, the gas supply in the model is not sufficient to fulfill the future gas demand of the locations. To fulfill this future demand, the model should invest in either gas storage or power-to-gas units. In experiments with less time steps, it is probably quickly found that these investments should not be made, as they are not required: the way that we modeled gas supply allows it to be fed into the model "for free". Although the instances with 11 and 12 time steps do not seem very different at first sight, the latter one is more complicated for the solver due to a difficult *trade-off* that the smaller cases do not have. When running instances with 17 time periods but less locations, we saw a similar pattern: when the gas supply was not sufficient for each time period, the solver would not solve to optimality anymore.

Running experiments with the smaller cases provided us with some insight in the predictability of running times for solving the MESDP:

- Difficult trade-offs seem to have a larger influence on the solver than merely the problem sizes. Therefore, for our experiments, problem sizes are not very useful for predicting running times;
- When certain difficult trade-offs for the solver can be figured out beforehand (for instance, the required investments to fulfill gas demand), one should account for the pos-

sibility that the solver will not solve to optimality within a reasonable time limit, as these trade-offs have a highly complicating effect on the solver.



(a) The case with 11 time steps. This instance solved to optimality in less than 300 seconds. In this instance, it is not required to invest in storage assets.

(b) The case with 12 time steps. This instance did not solve to optimality after two days of running. In this instance, the model is "forced" to invest in storage assets.

Figure 7.2: Investments made over time for the regular instance with 7 locations and 11 and 12 time steps, respectively

We want to highlight here that the latter is in line with our complexity analysis: as the strong NP-hardness of the MESDP is caused by the possibility of loading different assets on the same arc (see Chapter 3), we can expect that forcing investments in a larger variety of assets (versus cases where some assets, such as storage, can be discarded) is challenging for the solver.

Experiments with constraint elimination By running experiments with the regular instance of the MESDP with the exclusion of some constraints from the model (3.1), we investigate whether it might be possible to highlight the "most difficult" constraints. For this, we have considered the following variations:

- Move the capacity factor u from constraints (3.1b), (3.1d) and (3.1k) to the objective function (3.1). This results in a different step cost function, and could possibly be beneficial for the solver.
- Exclude the possibility of building storage, i.e., remove the W variables from the mass balance constraint (3.1a) and remove storage constraints (3.1k) and (3.1l).
- Set all the conversion-, loss- and storage factors (μ_{ij}, η_k and ξ_k) equal to 1.
- Remove the constraints representing the equal amount of heat- and electricity supplied by the CHP units, i.e., constraints (3.1h) and(3.1i).

Eliminating the above constraints alternately did not have a significant effect on the efficiency of the solver. In fact, in some cases, removing one of the above constraints actually increased the time that the solver required to solve an instance. Highlighting the most difficult constraints can apparently not be done in such a straightforward manner and it requires, at the least, a more extensive experiment than this one. In addition to this, we are mainly interested in applying the model with all of the constraints as listed in Chapter 3 and might get more insight in that specific problem by comparing different instances under the complete set of constraints.

Investigating in more detail the influence of the constraints on the solver's efficiency therefore lies beyond the scope of this research.

Constructing the set of test instances We now generate instances for the remainder of this study. As we have a bit more insight in properties of instances that have an impact on the difficulty, we try to create a diverse set of instances with respect to these properties, while we try to avoid bias and above all generate realistic instances[2]. For instance, by having more knowledge on which trade-offs have a large influence on the solver, we change the data input of some instances in such a way that we expect these trade-offs to be easier or more difficult (for instance, by smaller differences in the price or capacity of different assets). We also vary the demand profiles and the gas supply, as these are expected to have a large influence on the solutions, and look into instances with more restrictions on building assets. Lastly, we also examine results when the current network that we derived before is taken into account.

We then arrive at the set of instances that is provided in Table 7.1.

Instance	Name	Difference with regular instance
Regular instance	7reg, 28reg	-
Instance with demand variation 1	7dem1, 28dem1	Demand is the same for each energy carrier
Instance with demand variation 2	7dem2, 28dem2	Demand is the same in each time period
Instance with cost/capacity variation	7cc, 28cc	P2G and heat network and -storage are cheaper
Instance with gas supply variation	7gas, 28gas	Less gas supply over the whole time period
Instance with updated data	7ud, 28ud	Smaller CHP units, restrictions on building supply, gas supply in less locations
Instance with Brownfield data	7bf, 28bf	A start network (at t= -1) is taken into account

Table 7.1: Variations on the regular instance with 7 or 28 locations and 17 time periods

We realize that we might still be limited in assessing implementations based solely on this test set: it is not a particularly large test set, which makes it difficult to generalize our findings, and the set might not be representative for other cities, as it only uses (slightly adjusted) data of the city of Eindhoven. Still, we think that for our purpose in this research, i.e., highlighting difficult parts of the MESDP and potentially improving parameter settings of the solver and/or find other good solution methods, this test set should suffice due to the variety in its input data.

7.2.1. Experiments with default settings

In the first empirical experiment we intend to:

1. find out which parts of the model are difficult for the solver by running various instances with default settings

Inspecting the results of implementations with smaller instances and less constraints allowed us to create a diverse set of test instances in the previous section. We now assess the performance of the solver on this set of instances, based on various measures of efficiency.

Time to optimality For all of the instances in Table 7.1, we found that the solver cannot solve

[2]We emphasize again that we are most interested in using the MESDP for solving practical problem cases, and therefore, we do not investigate instances that have highly unlikely input data with respect to the energy scenarios.

the problem optimally within multiple hours of running. We tried whether optimality would be achieved within several days for a subset of instances, but we were not able to solve a single case from this subset to provable optimality. To handle the assessment of the set of instances, we therefore focus on finding solutions of an acceptable quality, i.e., focus on solving the problem with a small optimality gap.

For the 28-location case, even solving the LP relaxation is challenging, requiring over one hour for some of the instances. For this reason, we change the method with which the problem in the root node is solved for the larger cases.

Solving the root node problem We conduct experiments with Gurobi's primal simplex method, dual simplex method and barrier method for solving the root node problem for the set of instances in Table 7.1. Figure 7.3 provides a visualization of these results. It turns out that switching to the barrier method for the 28-location cases has a significant effect on the time required to solve the LP relaxation (and hence, on the total solution time). For the 7-location case, although the impact of switching between different solution methods was less significant (they all required less than 1 or 2 seconds for each instance), the barrier method also appeared to be slightly faster. Therefore, for the remainder of this research, we solve the problem in the root node using the barrier method for all of the experiments.

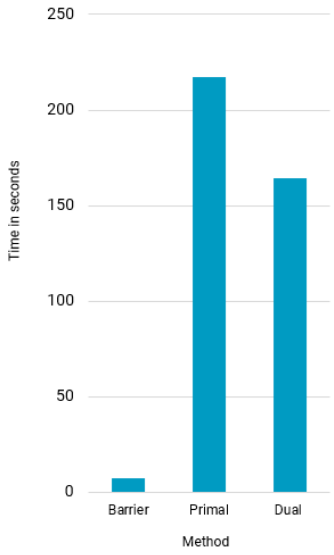


Figure 7.3: Average amount of time required for solving the LP relaxation in the root node using the primal/dual simplex method and the barrier method, taken over all instances in Table 7.1.

Time to solve to a fixed gap To get more insight in the time the solver requires to come up with good quality solutions, we have let the solver run until a solution with an optimality gap of less than 7% was found. We found that the solver, in almost all cases, quickly arrives at this point. We see something different happening when we set the fixed gap to 2.5%, as this requires significantly more time in most cases. Table 7.2 provides an overview of the wall clock time after which the solver is able to reach a gap of 7% and 2.5%.

	Solving to a gap of <7%			Solving to a gap of <2.5%		
instance	nodes explored	performed iterations	time	nodes explored	performed iterations	time
7reg	96	77398	16.21	21755	5973316	1073.48
7dem1	1	18833	4.48	1	28836	8.36
7dem2	1	12562	2.54	1	17311	4.33
7cc	1001	384639	106.71	108880*	58061968*	7200.09*
7gas	1	11980	1.93	112	40697	10.78
7ud	1	19024	3.9	1185	191279	57.08
7bf	108	44860	10.21	324796	24970327	2472.27
28reg	1	320378	1054.82	1*	1801477*	7200.08*
28dem1	1	33659	36.8	1	416104	1961.35
28dem2	1	9123	46.44	1	1183994	6212.60
28cc	1	841818	3625.18	1*	1599912*	7200.04*
28gas	1	5913	17.56	1	122943	205.75
28ud	1	35492	29.22	1	191604	233.29
28bf	1	195443	440.14	1	195443	460.85

Table 7.2: Time required to solve with optimality gaps of 7% and 2.5% . Also provided are the number of explored nodes and the number of performed simplex iterations at these respective times. For these experiments, we have set a time limit of 7200s. Results with an asterisks did not reach a gap of 2.5% within this time limit.

Investigating the output of the solver in more detail, it appears that after a while there is almost always very little progress in the best bound and no progress at all in the incumbent, resulting in an optimality gap that decreases extremely slowly. Table 7.3 provides the output log of a case where we let the solver run for over twelve hours. These kind of logs could indicate a lack of progress in the best node, a pattern that we see occurring after a while in all of the instances. In the next section, we apply different methods that could improve the progress in the best node.

	Nodes		Current Node			Objective Bounds			Work	
	Expl	Unexpl	Obj	Depth	IntInf	Incumbent	BestBd	Gap	It/Node	Time
	0	0	5.914400E+07	0	52	4.245500E+08	5.914400E+07	86.10%	-	0s
H	0	0				8.825821E+07	5.914400E+07	33.00%	-	0s
H	0	0				7.502687E+07	5.914400E+07	21.20%	-	0s
	0	0	6.490200E+07	0	123	7.502700E+07	6.490200E+07	13.50%	-	1s
	0	0	6.513700E+07	0	153	7.502700E+07	6.513700E+07	13.20%	-	1s
	0	0	6.513700E+07	0	153	7.502700E+07	6.513700E+07	13.20%	-	1s
	0	0	6.526100E+07	0	133	7.502700E+07	6.526100E+07	13.00%	-	1s
	0	0	6.527700E+07	0	149	7.502700E+07	6.527700E+07	13.00%	-	2s
	0	0	6.527700E+07	0	152	7.502700E+07	6.527700E+07	13.00%	-	2s
	0	0	6.541100E+07	0	144	7.502700E+07	6.541100E+07	12.80%	-	2s
	0	0	6.541300E+07	0	144	7.502700E+07	6.541300E+07	12.80%	-	2s
	0	0	6.557700E+07	0	138	7.502700E+07	6.557700E+07	12.60%	-	2s
H	0	0				7.222468E+07	6.557700E+07	9.20%	-	2s
						...				
	68948	35268	6.733000E+07	56	94	6.989000E+07	6.707100E+07	4.03%	243	2863s
	69190	35448	cutoff	60		6.989000E+07	6.707200E+07	4.03%	243	2871s
	69456	35573	cutoff	79		6.989000E+07	6.707300E+07	4.03%	242	2878s
	69652	35685	cutoff	89		6.989000E+07	6.707400E+07	4.03%	242	2884s
	69887	35822	6.881900E+07	63	74	6.989000E+07	6.707500E+07	4.03%	242	2892s
H	70051	35502	6.985568E+07				6.707500E+07	3.98%	242	2892s
	70203	35616	6.867000E+07	62	56	**6.985600E+07**	6.707700E+07	3.98%	241	2900s
	70368	35718	6.849600E+07	67	80	**6.985600E+07**	6.707800E+07	3.98%	241	2909s
						...				
	1450021	632647	6.914200E+07	54	58	**6.985600E+07**	6.850300E+07	1.94%	165	44174s
	1450241	632715	cutoff	70		**6.985600E+07**	6.850400E+07	1.94%	165	44183s
	1450508	632786	cutoff	102		**6.985600E+07**	6.850400E+07	1.94%	165	44193s
	1450683	632826	cutoff	65		**6.985600E+07**	6.850400E+07	1.94%	165	44204s
	1450913	632919	6.908400E+07	77	44	**6.985600E+07**	6.850400E+07	1.93%	165	44215s
	1451142	633029	cutoff	60		**6.985600E+07**	6.850400E+07	1.93%	165	44225s

Table 7.3: Output log for the regular instance with 7 locations and 12 time periods. After 2900 seconds, the value of the incumbent objective bound (bold) remains the same and there is very slow progress in the best lower bound.

7.2.2. Experiments with parameter variations and valid inequalities

Before arriving at the main results of the experiments with the Gurobi solver under the parameter variations and the implementation of valid inequalities (VI), we must first motivate the choices we made during the implementation.

Implementing valid inequalities As mentioned in Chapter 5, there are multiple ways in which VI can be implemented in a MIP solver. To find out which is the best, we run a few experiments with VI on a smaller test case, on which we shall base our strategy for the experiments with the entire test set. To examine which is the best way for the entire set of test cases, is a point for further research.

For implementing the VI, we test the following options on the test case:

- No VI;
- Implementing the VI a priori (as regular constraints);
- Implementing the VI through callback functions [3] ;
- Implementing the VI a priori as lazy constraints [4].

The results of these experiments are provided in Table 7.4.

		Optimality after	Nodes explored	Gap
Default settings	No VI	140.66	16474	-
	VI a priori as regular constraints	-	1736	8.88%
	VI added as cuts through callback	306.50 seconds	1371	-
	VI a priori as lazy constraints	53.80 seconds	6303	-
Solver heuristics and cuts off	No VI	-	298970	4.45%
	VI a priori as regular constraints		225808	4.99%
	VI added as cuts through callback	-	1638	16.73%
	VI a priori as lazy constraints	-	476372	4.47%

Table 7.4: Results for the test case, 7reg with 5 locations and 13 time periods, after 350s of running. To get more insight into the effect of the VI, we have also run an experiment with the solver cuts and heuristics off.

It follows from Table 7.4 that implementing the VI a priori as lazy constraints is the preferred strategy, as the solver finds the optimal solution within a significantly shorter amount of time than when VI are not implemented. For the remainder of this research, we refer to this strategy when we mention the implementation of the VI.

Comparative experimentation We now provide the results of the experiments with the default method (i.e., the solver's default parameter settings), the four parameter variations as

[3] It is important to note that we apply the callback functions in every node here, which might not be the most practical way, as the processing time of each node increases heavily. An alternative would be to only apply it for the first few nodes, or only every k nodes for a chosen number k. However, we do not consider this in this research.

[4] As mentioned in Chapter 5, for the lazy constraint strategy, we should implement the VI as *aggressive* lazy constraints. In that way, they are added to the model when they are violated in the LP relaxations of the branch and bound method. Note that VI, by definition, are not violated in integer feasible solutions, which is why this is necessary.

introduced in Chapter 5 and the valid inequalities, respectively. These experiments are all conducted for each of the 14 instances in Table 7.1 and for a single instance of the 110 node case, resulting in a total of $6 \cdot 15 = 90$ experiments. In each these experiments, we let the solver run for 7200 seconds and we solve the LP relaxation in the root node using the barrier method.

Table 7.5 provides an overview of the various measures of efficiency for the experiments with parameter variation and VI. In the table, "default" refers to Gurobi's default settings; "very strong cuts" refers to setting the Gurobi parameter `Cuts = 3`; "strong flow cover cuts" refers to setting the Gurobi parameter `FlowCoverCuts = 2`; "focus on best bound" refers to setting the Gurobi parameter `MIPFocus =3`; "strong branching" refers to setting the Gurobi parameter `VarBranch`= 3; and "VI" refers to our implementation of the valid inequalities from Chapter 5. The measures of efficiency explored in this table are the best optimality gap, the gap integral value, the best incumbent value, the number of nodes explored, the total number of iterations and the average amount of iterations per node, all upon termination of the algorithm. To provide insights into the entire solution process, we provide visualizations of the evolution of the lower- and upper bound and the gap function for each of the experiments in the table. We choose to highlight the evolution of the bounds and the gap function for two remarkable instances here, and refer to Appendix A for visualizations of all the experiments in Table 7.5. To this end, Figure 7.4 and Figure 7.6 illustrate the evolution of the lower bounds and the gap functions of the experiments with the cc28, dem27 and dem228 case, respectively.

instance	method	measure of efficiency					
		gap	gap integral	best incumbent	nodes explored	total iterations	iterations / node
reg7	default	0.0201	179.2224	120180000	172470	63296490	367
	very strong cuts	0.0194	165.1325	120140000	86183	33266638	386
	strong flow cover cuts	0.0192	157.6974	120230000	146344	36439656	249
	focus on best bound	0.0218	171.244	120580000	79398	41048766	517
	strong branching	0.0205	160.7598	120090000	17927	4463823	249
	VI	0.0143	117.0956	**119960000**	190637	77779896	408
reg28	default	0.0635	491.5493	135440000	-	-	-
	very strong cuts	0.018	225.159	**129230000**	344	1433104	4166
	strong flow cover cuts	0.0235	307.9116	129916300	491	2591007	5277
	focus on best bound	0.0235	242.9735	130066400	195	834015	4277
	strong branching	0.0635	494.6315	135440000	-	-	-
	VI	0.0223	247.5892	129716400	1086	3991050	3675
ud7	default	0.0181	161.6179	**121980000**	145028	51339912	354
	very strong cuts	0.0218	165.0604	122210000	170211	65531235	385
	strong flow cover cuts	0.0184	151.6874	121990000	247174	70691764	286
	focus on best bound	0.0175	141.7685	122180000	107758	48275584	448
	strong branching	0.0214	161.2256	122070000	28703	6372066	222

	VI	0.0173	131.6593	123630000	298691	90204682	302
ud28	default	0.0221	213.2727	130100000	1134	3861270	3405
	very strong cuts	0.0196	243.4107	**129790000**	3252	4627596	1423
	strong flow cover cuts	0.0258	279.4356	130620000	566	2335882	4127
	focus on best bound	0.0215	299.1863	130193700	357	862155	2415
	strong branching	0.0278	226.5813	130930000	258	1097532	4254
	VI	0.0105	107.6149	133880000	3605	4070045	1129
gas7	default	0.0116	94.7764	**387580000**	183265	54063175	295
	very strong cuts	0.0147	106.6992	388770000	223344	69013296	309
	strong flow cover cuts	0.013	96.5711	388050000	140437	51680816	368
	focus on best bound	0.0159	116.0008	389400000	192116	62053468	323
	strong branching	0.0121	102.8963	387780000	34639	4606987	133
	VI	0.0128	95.3993	387940000	195516	75860208	388
gas28	default	0.0084	77.0103	385100000	2920	3906960	1338
	very strong cuts	0.0074	72.1316	384720000	1590	4044960	2544
	strong flow cover cuts	0.0056	69.987	**384060000**	2217	4265508	1924
	focus on best bound	0.0095	80.7574	385560000	1498	3282118	2191
	strong branching	0.0098	84.7147	385730000	504	1143072	2268
	VI	0.008	75.6933	385080000	5211	5455917	1047
dem17	default	0.013	105.7939	260080000	149208	45060816	302
	very strong cuts	0.0164	123.0294	260770000	111586	38385584	344
	strong flow cover cuts	0.0118	99.459	**259650000**	146618	42519220	290
	focus on best bound	0.015	113.0677	260440000	87176	38531792	442
	strong branching	0.0142	108.3681	260030000	13602	3441306	253
	VI	0.0118	93.9412	259840000	229405	70656740	308
dem128	default	0.0221	213.2727	248800000	1134	3861270	3405
	very strong cuts	0.0196	243.4107	249450000	3252	4627596	1423
	strong flow cover cuts	0.0258	279.4356	249710000	566	2335882	4127
	focus on best bound	0.0215	299.1863	**248287200**	357	862155	2415
	strong branching	0.0278	226.5813	248840000	258	1097532	4254
	VI	0.0142	121.9495	249240000	894	3449052	3858
dem27	default	0.0083	70.5892	453190000	146242	46212472	316
	very strong cuts	0.0111	81.4799	454380000	133367	60681985	455
	strong flow cover cuts	0.009	68.9056	453610000	157990	51504740	326
	focus on best bound	0.0109	80.4669	454360000	160800	49687200	309
	strong branching	0.0115	85.2057	454400000	17205	4989450	290
	VI	0.008	63.8902	**453110000**	149392	63192816	423
dem228	default	0.0169	171.5848	455170000	-	-	-
	very strong cuts	0.0207	152.4659	456800000	-	-	-

	strong flow cover cuts	0.0344	248.0402	462930000	-	-	-
	focus on best bound	0.0154	157.3556	**454570000**	3	22314	7438
	strong branching	0.0169	173.7692	455170000	-	-	-
	VI	0.0234	188.6067	458110000	1	7835	7835
cc7	default	0.0506	379.6645	102420000	153555	58197345	379
	very strong cuts	0.0544	403.2511	102720000	116470	60680870	521
	strong flow cover cuts	0.0525	393.463	102660000	141323	61051536	432
	focus on best bound	0.0528	391.8599	102590000	119652	56356092	471
	strong branching	0.0469	353.8059	101700000	16912	7813344	462
	VI	0.0372	298.1962	**101240000**	149018	55434696	372
cc28	default	0.0515	3096.721	110030000	-	-	-
	very strong cuts	0.073	636.7262	112630000	-	-	-
	strong flow cover cuts	0.0414	484.2601	**108910000**	-	-	-
	focus on best bound	0.079	691.3855	113460000	-	-	-
	strong branching	0.0516	3170.866	110030000	-	-	-
	VI	0.0572	356.1199	110730000	-	-	-
bf7	default	0.0225	165.1889	96568000	916504	57556451	62.8
	very strong cuts	0.0239	175.618	96707000	755132	71057921	94.1
	strong flow cover cuts	0.0222	163.0133	96614000	1355035	64906177	47.9
	focus on best bound	0.0226	166.8726	96613000	712904	53253929	74.7
	strong branching	0.0222	167.966	96576000	89072	7312811	82.1
	VI	0.022	162.8896	**96568000**	886765	57373696	64.7
bf28	default	0.0069	116.7492	119710000	5448	1786944	328
	very strong cuts	0.0067	73.4735	**119590000**	3677	3110742	846
	strong flow cover cuts	0.0072	70.8367	119750000	5306	2032198	383
	focus on best bound	0.0079	79.6414	119660000	3805	4714395	1239
	strong branching	0.0033	50.0915	120050000	535	613110	1146
	VI	0.0069	90.0359	119720000	9937	7343443	739
reg110	default	0.112	837.51	147100000	350	389900	1114
	very strong cuts	0.108	641.796	146470000	0	-	-
	strong flow cover cuts*	0.0748	692.3828	**141320000**	615	586095	953
	focus on best bound*	0.103	445.988	145760000	1	14013	14013
	strong branching	0.0898	796.0312	143640000	57	115767	2031
	VI*	0.142	872.033	152630000	23	182988	7956

Table 7.5: Results for the implementations with parameter variations and the addition of valid inequalities, respectively. The best incumbent values for each instance are bold. In instances with an asterisks the solver ran out of memory before reaching the time limit.

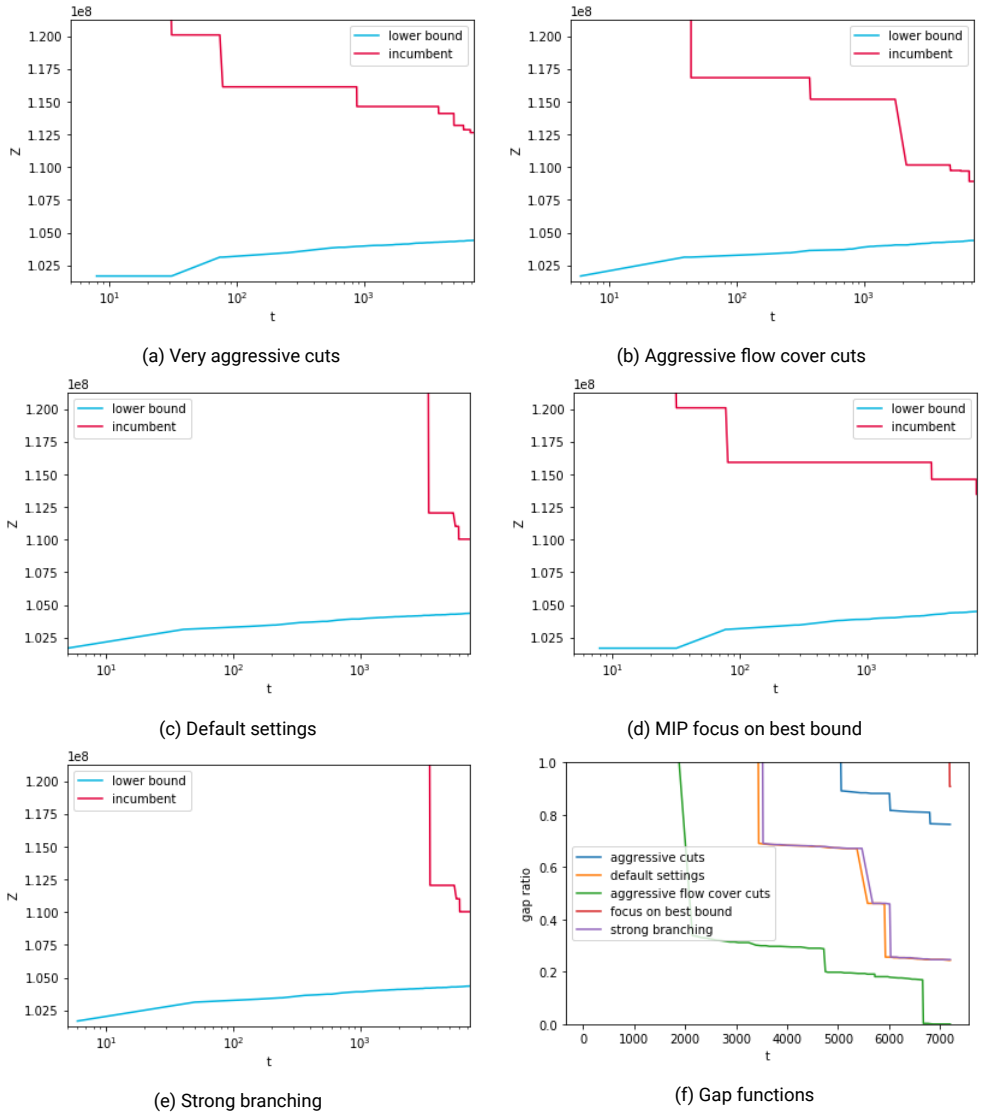
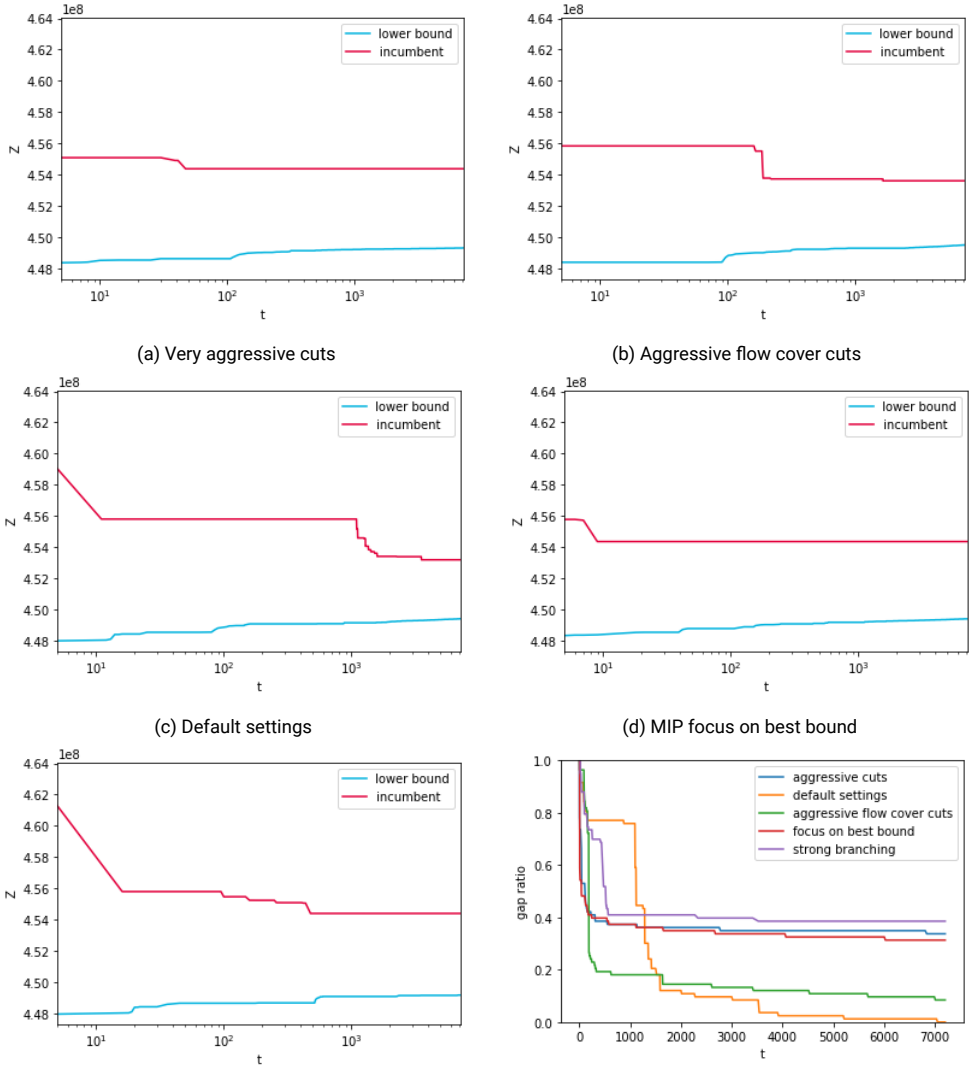


Figure 7.4: Graphs corresponding to the evolution of the bounds for the instance of cc28 and their corresponding gap functions



(a) Very aggressive cuts

(b) Aggressive flow cover cuts

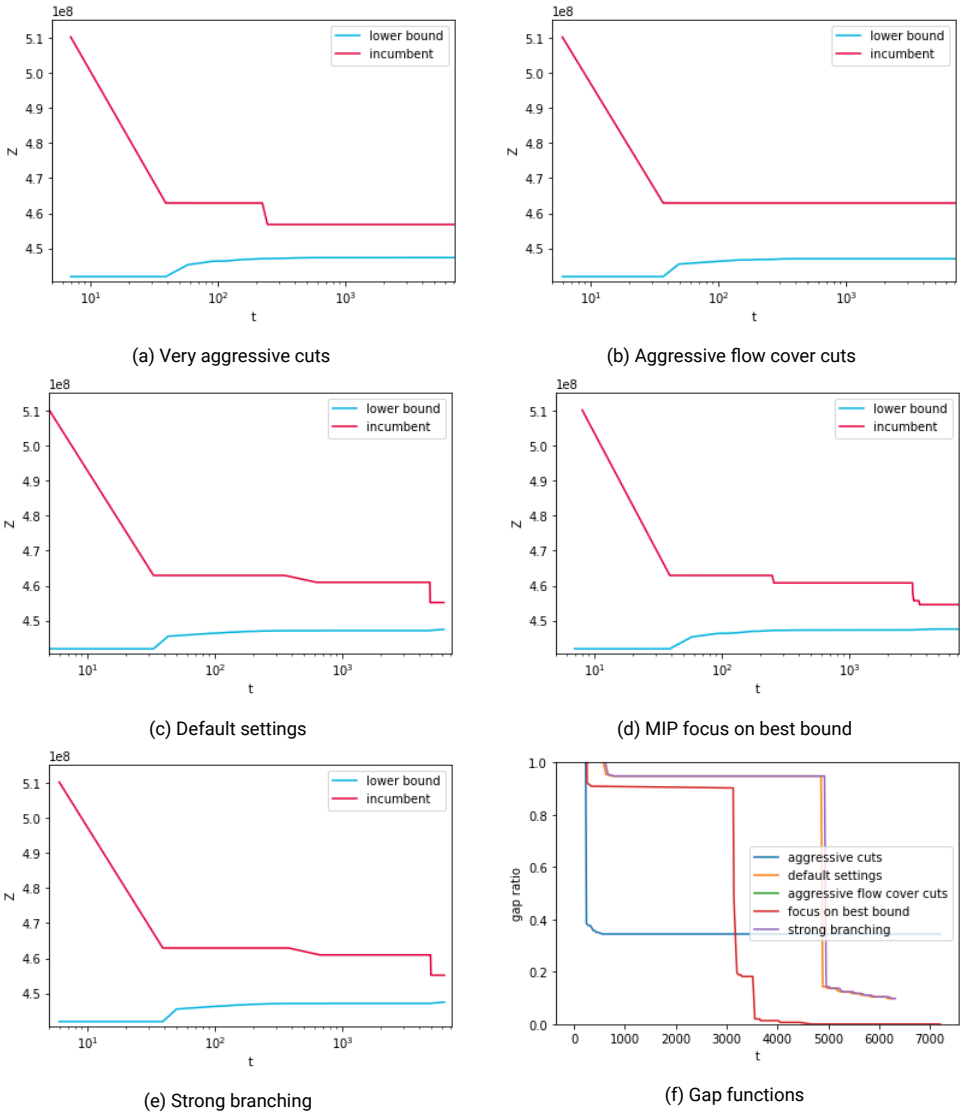
(c) Default settings

(d) MIP focus on best bound

(e) Strong branching

(f) Gap functions

Figure 7.5: Graphs corresponding to the evolution of the bounds for the instance of dem27 and their corresponding gap functions



(a) Very aggressive cuts

(b) Aggressive flow cover cuts

(c) Default settings

(d) MIP focus on best bound

(e) Strong branching

(f) Gap functions

Figure 7.6: Graphs corresponding to the evolution of the bounds for the instance of dem228 and their corresponding gap functions

Main findings

When assessing the effectiveness of algorithms, there is always a danger of over-generalizing the results. Our main findings are all based on the experiments that terminated after 7200 seconds. Our ability to generalize these findings is therefore limited, as we do not know the actual optimal objectives for the experiments. We therefore emphasize that any statements regarding the effectiveness of the investigated methods describe their effectiveness *within the time limit*. In addition, some parameters, such as the best incumbent value and the best optimality gap, are found in the last time period. Marking a method as “best” based solely on such measures, might not be a good idea. Consider for instance the evolution of the bounds of 28cc, as illustrated in Figure 7.4. If the time limit would have been set to 100 seconds, then the default

method would have a very large optimality gap, whereas it has the second-best gap upon termination of the algorithm. The MIP focus parameter setting, on the other hand, has a smaller gap at 100 seconds, but a much larger gap upon termination of the algorithm. In cases where the solver is only allowed to run for a short amount of time (<1000 seconds), the preferred parameter settings might differ from the ones that we introduce here. In our comparison of the results, we explicitly focus on the performance of the methods within the time limit of 7200 seconds.

We now provide some remarkable findings from Table 7.5.

General performance First and foremost, what stands out in in the table is that the Gurobi solver generally seems to perform quite well (both with and without parameter variations and VI) for the investigated set of problem instances within the time limit of 7200 seconds, resulting in an average optimality gap of 2.84% over all cases and an average gap of 2.29% for the set of 7- and 28-node cases. This motivates the use of the Gurobi solver for solving practical problem instances of the MESDP.

Another remarkable finding is that, in many of the experiments, no significant differences were found between the best incumbent values of the different methods. For all of the experiments with valid inequalities and parameter settings variations, it holds that in 61 of the 75 experiments the best incumbent value deviated less than 1% of the best incumbent value of the default method. The biggest derivation of the incumbent values is found within the 28reg problem instance, where the cuts-, flow cover cuts- and bound focus parameter and the VI all outperformed the default method, resulting in incumbent values that are respectively 4.59%, 4.08%, 3.97%, 4.23% lower than the incumbent value of the default.

Comparing instances When comparing the same instances for different sizes, we see that the solver often finds small gaps for both the 7- and 28- node variations within the time limit. The gap integrals however do differ significantly, as the solver often finds a good optimality gap for the 7 node cases within only a few seconds, whereas the convergence of the gap in the 28 node cases is much slower. This phenomenon is illustrated for the dem27 and dem228 case in Figure 7.5 and 7.6. We see similar behaviour for the other cases, as illustrated in the figures in Appendix A. The large gap integral for the 28 node cases is caused by the poor quality of the feasible solutions, as the lower bound function of the 28 node cases does increase at a higher rate. The better lower bound function for the 28 node cases can be explained by the quality of the LP relaxations: the finer network has tighter LP relaxations and hence, tighter lower bounds.

Brownfield One of the research sub-questions was whether the addition of a current network would have a positive effect on the solver. To this end, we compare the optimality gaps of the different methods applied to the regular case, both with (“brownfield”) and without (“greenfield”) a starting network. The results are provided in Table 7.6. These show that for the 7 node case, the gaps for the brownfield case are not better than those for the greenfield case. For the 28 node case however, the optimality gaps under brownfield are significantly smaller for all methods, suggesting that a starting network makes it much simpler to solve the model

size	method	greenfield	brownfield
7 nodes	default	2.01%	2.25%
	cuts	1.94%	2.39%
	FCC	1.92%	2.22%
	MIPfocus	2.18%	2.26%
	varbranch	2.05%	2.22%
	user cuts	1.43%	2.20%
28 nodes	default	6.35%	0.69%
	cuts	1.80%	0.67%
	FCC	2.35%	0.72%
	MIPfocus	2.35%	0.79%
	varbranch	6.35%	0.33%
	user cuts	2.23%	0.69%

Table 7.6: The optimality gaps of the different methods applied to the regular case with greenfield and brownfield data

for the finer networks. The smaller degrees of freedom of the starting network combined with the tight LP relaxations for the 28-node cases result in an efficient solution procedure. This implies that brownfield data possibly has a positive effect on the solver, especially in the finer networks. However, as we only investigate the effect of brownfield data for a single case, caution must be applied in drawing broad conclusions.

Trade-offs Apart from the 110 node case, the instances 7cc and 28cc are the most difficult problem instances, resulting in the largest gap and gap integrals over all. These instances model a situation in which the prices for assets lie closer together than in other instances. We therefore observe that a difficult trade-off between building different assets is a good predictor of the difficulty of an instance.

Useful tricks We observe that in some experiments, the solver found good solutions without exploring any nodes in the branch and bound tree. We assume that the model therefore benefits significantly from many of Gurobi's additional tricks, such as the presolve methods and cutting planes.

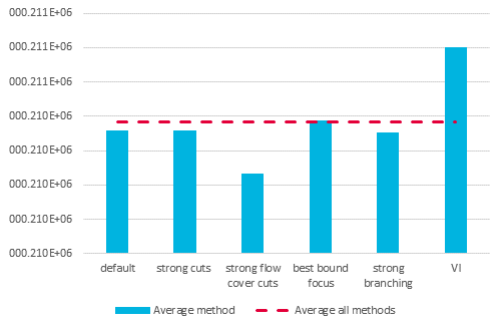
Overall performance To assess which method performs best overall, we examine the following measures of efficiency in more detail for the experiments in Table 7.5: the optimality gap, gap integral and incumbent values. The combination of these measures of efficiency provides insight a good base to compare the different methods on (Maher et al., 2019). Tables 7.7 and 7.8 summarize the results of these 90 experiments. In Table 7.7, the total number of instances for which each method outperformed the others based on these methods is given. Considering only the "winner" of each experiment is however not very insightful, as we are mainly interested in each method's performance over the entire set of instances. Therefore, in Table 7.8, the average values of the gap, gap integral and incumbent over all of the experiments is given for each method. This is also visualized in Figure 7.7.

	Best gap	Best gap integral	Best incumbent
default	1	1	3
very strong cuts	1	2	**4**
strong flow cover cuts	3	1	3
focus on best bound	1	0	1
strong branching	1	1	0
VI	**7**	**9**	3

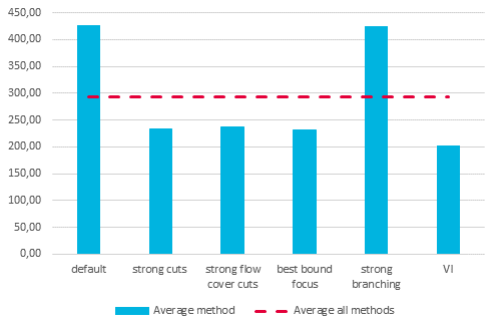
Table 7.7: The number of instances in which each method resulted in the best gap, the best gap integral value and the best incumbent value after 7200 seconds of running.

	Average gap	Average gap integral	Average incumbent
default	2.98%	424.97	2.10317E+08
very strong cuts	2.90 %	233.92	2.10314E+08
strong flow cover cuts	**2.56** %	237.54	**2.10062E+08**
focus on best bound	2.92 %	**231.85**	2.10375E+08
strong branching	2.93 %	424.23	2.10304E+08
VI	2.72%	201.51	2.10817E+08

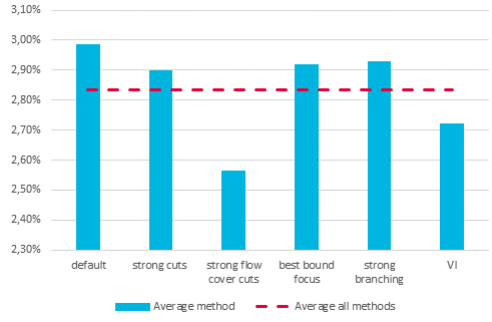
Table 7.8: Average values of the best gap, the best gap integral value and incumbent for each method after 7200 seconds.



(a) Best incumbent value



(b) Best gap integral



(c) Best gap

Figure 7.7: Average values over the set of instances of the best incumbent value, best gap integral and best gaps. The bars denote the average for each method, and the dotted line denotes the average over all methods.

The results of Table 7.7 and Table 7.8 show that, over all, the cut-based parameters and the valid inequalities perform best. The flow cover cuts resulted in the best average incumbent over all experiments. In addition, for the most difficult instances, cc28 and reg110, we observe in Table 7.5 that the flow cover cuts parameter settings outperformed the other methods and improved the default method with 1.02% and 3.93%, respectively.

To also examine the performance of the methods on the smaller and larger cases, respectively, we categorize the results by their size in Table 7.9 and Table 7.10. The 110 node case has a large effect on the overall performance of the methods and it is therefore also interesting to compare the performance without this particular difficult case. For instance, the VI method appears much better without considering its bad performance for the 110 node case (for which the PC ran out of memory before reaching the time limit). As demonstrated in Table 7.10, the VI outperform other methods on all three measures of efficiency for each of the 28-node cases and on the best gap and best gap integral in the 7-node case.

	Best gap	Best gap integral	Best incumbent
default	2.06%	165.2647429	**2.2029E+08**
strong cuts	2.31%	174.3243571	2.2081E+08
strong flow cover cuts	2.09%	161.5424	2.2040E+08
best bound focus	2.24%	168.7543429	2.2088E+08
strong branching	2.13%	162.8896286	2.2038E+08
VI	**1.76**%	**137.5816286**	2.2033E+08

Table 7.9: Average values of the best gap, the best gap integral value and incumbent for each method after 7200 seconds of running for experiments for the set of 7 node cases

	Best gap	Best gap integral	Best incumbent
default	2.73%	625.7371143	2.0938E+08
strong cuts	2.36%	235.2539429	2.0894E+08
strong flow cover cuts	2.34%	248.5581143	2.0954E+08
best bound focus	2.55%	264.3551429	2.0910E+08
strong branching	2.87%	632.4621571	2.0975E+08
VI	**1.98**%	**157.3768714**	**2.0851E+08**

Table 7.10: Average values of the best gap, the best gap integral value and incumbent for each method after 7200 seconds of running for experiments for the set of 28 node cases

For a graphical representation of a 7-, 28- and 110-node network that were generated by the solver in one of the experiments, we refer to Figure A.21, A.22 and A.23, respectively.

7.3. Decomposition methods

In this section, we provide the results of the Benders decomposition (BD) and Lagrangean relaxation (LR) method for the MESDP, respectively.

7.3.1. Benders decomposition

We implemented the BD method according to Algorithm 2. First, we solve the PHASE 1 problem. Because the computing times for the PHASE 1 problem were quite extensive, we decided

to set a maximum of 100 iterations for the for loop in PHASE 1 and terminate it once the objective has not improved for 30 iterations. Second, we solve the PHASE 2 problem, letting the solver run for a maximum of 7200 seconds. The results of these experiments for the set of 14 instances is provided in Table 7.11. The evolution of the corresponding lower bounds that were generated during PHASE 2 are visualized in Appendix A. For the 110 node case, the solver ran out of memory before completing PHASE 1.

Instance	Number of active Benders cuts	Total iterations	Nodes explored	Best bound	Best incumbent	Gap
reg7	3268	33234	8357	1.09E+08	-	-
reg28	238	2912	27	6.66E+07	-	-
ud7	2662	26311	5111	1.06E+08	-	-
ud28	357	3410	1	6.30E+07	-	-
gas7	4327	21140	4302	3.695206E+08	4.3694571E+08	15.43%
gas28	255	2665	39	3.54E+08	-	-
dem17	1615	28307	4905	243644445.8	-	-
dem128	340	2177	1	217262166.0	-	-
dem27	3946	40945		4.446179E+08	5.62E+08	20.90%
dem228	272	3808	299	4.17E+08	-	-
cc7	2992	27724	3846	9.19E+07	-	-
cc28	340	3148	1	8.07E+07	-	-
bf7	250	15232	3333	8.94E+07	-	-
bf28	544	2251	1	9.53E+07	-	-

Table 7.11: Results for the Benders decomposition algorithm for the set of instances

Our implementation of the BD algorithm turns out to be disappointingly slow. In only two experiments it succeeded in finding an incumbent value. Also, in all of the experiments, the best obtained lower bound deviated significantly from the best lower bound that was obtained by the solver's default branch-and-cut method.

We believe that a large part of the computing times for both of the BD can be accounted for by the time it takes for the algorithm to initialize and solve the models in each iteration. For BD, each time the solver finds a new incumbent, 17 (the number of time periods) disaggregated Benders sub-problems have to be initialized and solved to optimality. Although these problems could all be solved in a few seconds, this is overall still a time-consuming process. Specifically, it appears the solver takes a significant amount of time for the initialization phase. Coding the decomposition method in a different way, in which the solver can adjust model parameters instead of initializing an entire model in each iteration, might provide better results. We were restricted to the use of Pyomo in this research, and found that it was unfortunately not possible to implement BD without the full initialization of the submodels in each iteration.

Increasing the runtime for the BD as implemented here is not desirable, as the processing time of each node in the BD increases with the number of generated cuts (and this processing time is already quite excessive upon termination of the algorithm). A better strategy for improving the BD would be the implementation of additional computational enhancements, such as a tighter cut selection or a better initialized relaxation. In addition, the lack of optimality cuts in our problem might make BD as a solution method less attractive, as feasibility cuts are less

instance	best LR bound	best lower bound default
reg7	3.26E+06	1.18E+08
ud7	1.49E+06	1.20E+08
gas7	1.05E+07	3.83E+08
dem17	3.80E+06	2.57E+08
dem27	1.00E+07	4.49E+08
cc7	1.32E+06	9.72E+07
bf7	5.762E05.2241	9.32E+07

Table 7.12: The best lower bound derived by the Lagrangean relaxation versus the best lower bound derived by the default method for the 7 node instances.

effective in improving the bound than optimality cuts. So unless it is possible to derive tighter feasibility cuts, BD might not be an efficient method for solving the MESDP.

7.3.2. Lagrangean relaxation

We implemented the Lagrangean relaxation method according to Algorithm 3. We let the for loop continue for 1000 iterations and terminate when $\|\lambda^{p+1} - \lambda^{p}\| < 1E^{-6}$ or when a time limit of 7200 seconds is reached. The upper bound Z^* is derived by taking a known incumbent value for the problem instance, generated by letting running the solver for the original formulation for a few seconds. For the 28- node cases, the for loop terminated in only a few iterations, due to memory issues. Therefore the results pertaining to the 28-node cases will not be fully representative and will not be further investigated.

The results of the best bounds obtained by the Lagrangean relaxation method for the set of 7-node instances are provided in Table 7.12. In this table, we compare this lower bound to the best bound found through applying the default method to the original formulation.

Due to the very slow convergence of the LR function, as can be seen in the graphs in Appendix A, the LR method as we implemented it here is not useful.

The subgradient method has to initialize and solve a model in each iteration. As mentioned before, this takes quite some time and might (partly) cause the slow convergence of the LR function. Coding the algorithm in a different way might solve this. In addition, the convergence of the subgradient method of the LR might be improved by adjusting some of its parameters, or starting with a different incumbent value.

7.4. Summary

For the MESDP, the branch and cut algorithm of the MIP solver Gurobi was efficient for solving the set of problem instances. To optimize its performance, the following improvements had the best overall performance:

- Solve the initial LP relaxation using the solver's barrier method and;
- Implement the valid inequalities a priori as lazy constraints, or;

- Force the solver to generate more flow cover cuts.

Combinations of optimal parameter settings and valid inequalities might be efficient, but have not been tested in this research.

The Benders decomposition method and Lagrangean relaxation method as implemented in Algorithm 2 and Algorithm 3 respectively, both did not seem efficient for solving the MESDP.

We have not investigated all the ways in which the BD and the LR method could be improved for the MESDP. As we only implemented a rather basic version of both of these algorithms, the question whether these decomposition methods are efficient for solving the MESDP remains open.

8

Conclusions and recommendations

In this chapter, we conclude our research by answering the main research question and we provide a direction for future research.

8.1. Conclusions

In this thesis, we investigated the MESDP, a model that optimizes the design of a multi-energy network. As explained in Chapter 3.1, the goal of the model is to compute the optimal design investment plan for the energy system of the studied city, Eindhoven.

In Chapter 3 we introduced a network flow-based formulation for the MESDP and proved that it is strongly NP-hard. Most interested in practical applications of the model, we investigated the efficiency of a state-of-the-art MIP solver for solving the MESDP for a set of instances in Chaper 7. In addition, we developed two decomposition methods to examine whether either of these would provide a good strategy for solving the MESDP.

We now answer the main research question.

What are good solution methods for optimizing the design of multi-energy systems, using the developed model?

There are many possible strategies for solving (fixed charge) network design problems that are similar to the MESDP (Chapter 2). In this report, we focused on branch and bound methods with model-specific parameter settings and the addition of valid inequalities, as well as decomposition methods. In Chapter 3, we proved that the MESDP is strongly NP-hard, implying that it is (theoretically) highly difficult to solve. From a practical viewpoint, it appeared that state-of-the-art MIP solvers are able to solve large-sized instances of the MESDP to small optimality gaps. Implementing model-specific valid inequalities as well as forcing the solver to apply more flow cover cuts both had a positive impact on the solver. Lastly, the influence of taking into account brownfield data was particularly striking for the larger cases, implying that a starting network can significantly improve the quality of the solution for finer networks. Considering the fact that finer networks are able to model the real-life case more accurately, this fact is of interest for practitioners who are more interested in network expansion/brownfield optimization problems as opposed to greenfield network design problems.

8.2. Recommendations

As the theory of multi-energy systems is still rather new, there is much to explore in this field. From a network optimization perspective specifically, we believe that there is also much to explore in solving (variations of) the MESDP. We provide some ideas for future research, based on findings from this report.

In Chapter 7, we found that the MIP solver steadily provided good solutions for our set of instances. Because the MIP solver performed well for practical instances, we propose to continue the investigation into ways in which the MESDP can be solved more efficiently, using a state-of-the-art solver. We investigated the effect of parameter tuning and valid inequalities. Although there was some improvement, we were still not able to solve the larger cases to optimality. It would be interesting to find out whether this is achievable, possibly with a tighter (or different) formulation.

Unfortunately, we were not able to produce efficient decomposition algorithms for the MESDP in this report. Considering the decomposable structure of the MESDP, we still believe that decomposition methods could be effective. Therefore, we propose research in ways that the decomposition methods might be applied to the MESDP in an efficient manner, for instance with additional computational enhancements or with a different decomposition strategy. For Lagrangean relaxation, for instance, other relaxed constraints might result in better lower bounds. Also, the implementation of these algorithms in the Python API as opposed to Pyomo might be efficient.

One of the key findings in Chapter 7, were the good lower bound functions for the larger cases and the near-optimality solutions for the 28- node brownfield case. Experiments with heuristic methods could provide better upper bounds, that, combined with the good lower bound functions, might result in better solutions. Therefore, heuristic methods might be interesting for the MESDP.

Lastly, We hypothesize that the addition of a current network is beneficial for solving the model, and therefore suggest further experiments with more brownfield cases. Considering the computational benefits from adding a brownfield network, as well as the large amount of brownfield optimization problems that arise from the Energy Transition, these findings motivate a direction for future research on the MESDP/multi-energy systems optimization that incorporate current networks.

Bibliography

Adams, R. N. and Laughton, M. A. (1974). Optimal planning of power networks using mixed-integer programming. *PROC. IEE*, 121(2).

Ahuja, R., Magnantl, T. L., and Orlin, J. B. (1993). *Network flows Theory, algorithms and applictions*.

Balakrishnan, A., Li, G., and Mirchandani, P. (2017). Optimal network design with end-to-end service requirements. *Operations Research*, 65(3):729–750.

Barahona, F. (1996). Network Design Using Cut Inequalities. *Siam J. Optimization*, 6(3):823–837.

Bernard Fortz and Enrico Gorgone (2012). A Lagrangian Heuristic Algorithm for the Time-Dependent Combined Network Design and Routing Problem. *Networks*, 60(1):45–58.

Bertsekas, D. P. (1998). *Network Optimization: Continuous and Discrete Models*. Athena Scientific, Belmont, Massachusetts.

Bertsimas, D. and Tsitsiklis, J. (1998). *Introduction to Linear Optimization*. Athena Scientific.

Bruckner, T., Bashmakov, I. A., Mulugetta, Y., Chum, H., de la Vega Navarro, A., Edmonds, J., Faaij A., Fungtammasan B., Garg A., Hertwich E., Honnery, D., Infield, D., Kainuma, M., Khennas, S., Kim, S., Nimir, H. B., Riahi, K., Strachan, N., Wiser, R., and Zhang, X. (2014). Energy systems. *Climate Change 2014: Mitigation of Climate Change. Contribution of Working Group III to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change*.

Casisi, M., Pinamonti, P., and Reini, M. (2009). Optimal lay-out and operation of combined heat & power (CHP) distributed generation systems. *Energy*, 34(12):2175–2183.

Chang, M. S. (2014). A scenario-based mixed integer linear programming model for composite power system expansion planning with greenhouse gas emission controls. *Clean Technologies and Environmental Policy*, 16(6):1001–1014.

Chang, S.-G. and Gavish, B. (1995a). Lower Bounding Procedures for Multiperiod Telecommunications Network Expansion Problems. *Operations Research*, 43(1):43–57.

Chang, S.-G. and Gavish, B. (1995b). Lower Bounding Procedures for Multiperiod Telecommunications Network Expansion Problems. *Operations Research*, 43(1):43–57.

Conejo, A. J., Castillo, E., and García-bertrand, R. M. R. (2006). *Decomposition Techniques in Mathematical Programming*.

Costa, A. M. (2005). A survey on benders decomposition applied to fixed-charge network design problems. *Computers and Operations Research*, 32(6):1429–1450.

Dutta, A. and Lim, J.-i. (1992). A Multiperiod Capacity Planning Model for Backbone Computer Communication Networks. (January 2020).

Enexis (2019). Enexis Netbeheer Open data.

European Commission (2015). The Paris Protocol – A blueprint for tackling global climate change beyond 2020. *COMMUNICATION FROM THE COMMISSION TO THE EUROPEAN PARLIAMENT AND THE COUNCIL*.

Fisher, M. L. (2004). The lagrangian relaxation method for solving integer programming problems. *Management Science*, 50(12 SUPPL.):1861–1874.

Fragkos, I., Jans, R., and Cordeau, J. F. (2017). The Multi-Period Multi-Commodity Network Design Problem. *CIRRELT-2017-63*, (October).

Gabrel, V., Knippel, A., and Minoux, M. (1999). Exact solution of multicommodity network optimization problems with general step cost functions. *Operations Research Letters*, 25(1):15–23.

Gabrielli, P., Gazzani, M., Martelli, E., and Mazzotti, M. (2018). Optimal design of multi-energy systems with seasonal storage. *Applied Energy*, 219(October 2017):408–424.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman.

Gascon, V., Benchakroun, A., and Ferland, J. (1993). Electricity Distribution Planning Model: A Network Design Approach For Solving The Master Problem Of The Benders Decomposition Method. *INFOR: Information Systems and Operational Research*, 31(3):205–220.

Geidl, M., Koeppel, G., Favre-Perrod, P., Klöckl, B., Andersson, G., and Fröhlich, K. (2007). Energy hubs for the future. *IEEE Power and Energy Magazine*, 5(1):24–30.

Gendron, B. (2011). Decomposition methods for network design. *Procedia - Social and Behavioral Sciences*, 20:31–37.

Gendron, B. and Crainic, T. G. (1998). Multicommodity Capacitated Network Design.

Geoffrion, A. M. (1972). Generalized Benders Decomposition. *Journal of Optimization Theory and Applications*, 10(4):1162–1175.

Gurobi Optimization Inc. (2017). Algorithms I-Basics.

Gurobi Optimization Inc. (2019). Gurobi Optimizer Reference Manual. *Www.Gurobi.Com*, 6:572.

Held, M., Wolfe, P., and Crowder, H. P. (1974). Validation of subgradient optimization. *Mathematical Programming*, 6(1):62–88.

Holmberg, K. and Yuan, D. (1998). A Lagrangean approach to network design problems. *International Transactions in Operational Research*, 5(6):529–539.

Holmberg, K. and Yuan, D. (2000). A Lagrangian Heuristic Based Branch-and-Bound Approach for the Capacitated Network Design Problem. *Operations Research*, 48(3):461–481.

Hugo, A., Rutter, P., Pistikopoulos, S., Amorelli, A., and Zoia, G. (2005). Hydrogen infrastructure strategic planning using multi-objective optimization. *International Journal of Hydrogen Energy*, 30(15):1523–1534.

IEA (2019). IEA Sankey Diagram.

Kallrath, J. (2008). Modeling Difficult Optimization Problems. *Floudas C., Pardalos P. (eds) Encyclopedia of Optimization*.

Karp, R. M. (1975). On the Computational Complexity of Combinatorial Problems. *Networks*, 5:45–68.

Klotz, E. and Newman, A. (2013). Practical Guidelines for Solving Difficult Mixed Integer Linear Programs. *Surveys in Operations Research and Management Science*, 18(1-2):18–32.

Kroposki, B., Garrett, B., Macmillan, S., Rice, B., Komomua, C., Malley, M. O., Zimmerle, D., and Kroposki, B. (2012). Energy Systems Integration A Convergence of Ideas A Convergence of Ideas. *University college Dublin*, (July 2012):1–9.

Kubat, P. and Smith, J. M. G. (2001). A multi-period network design problem for cellular telecommunication systems. *European Journal of Operational Research*, 134(2):439–456.

Lawler, E. L. and Wood, D. E. (1966). Branch-and-Bound Methods : A Survey. 14(4):699–719.

Magnanti, T. L., Mirchandani, P., and Vachani, R. (1991). Modeling and Solving the Capacitated Network Loading Problem. *OR 239-91*, (January).

Magnanti, T. L., Mirchandani, P., and Vachani, R. (1993). The convex hull of two core capacitated network design problems. *Mathematical Programming*, 60(1-3):233–250.

Magnanti, T. L. and Wong, R. T. (1984). Network Design and Transportation Planning: Models and Algorithms. *Transportation Science*, 18(1):1–55.

Maher, S. J., Ralphs, T. K., and Shinano, Y. (2019). Assessing the Effectiveness of (Parallel) Branch-and-bound Algorithms. pages 1–49.

Mancarella, P. (2014). MES (multi-energy systems): An overview of concepts and evaluation models. *Energy*, 65:1–17.

Mancarella, P., Andersson, G., Peças-Lopes, J. A., and Bell, K. R. (2016). Modelling of integrated multi-energy systems: Drivers, requirements, and opportunities. *19th Power Systems Computation Conference, PSCC 2016*, (October 2017).

Matuschke, J., Skutella, M., Peis, B., and McCormick, T. (2014). Network flows and network design in theory and practice. (February).

McDaniel, D. and Devine, M. (1977). A Modified Benders' Partitioning Algorithm for Mixed Integer Programming. *Management Science*, 24(3).

Minoux, M. (1987). Network synthesis and dynamic network optimization. *North-Holland Mathematics Studies*, 132(C):283–323.

Omu, A., Choudhary, R., and Boies, A. (2013). Distributed energy resource system optimisation using mixed integer linear programming. *Energy Policy*, 61:249–266.

Orlin, J. and Nasrabadi, E. (2013). 15.053 Optimization Methods in Management Science. *Massachusetts Institute of Technology: MIT OpenCourseWare. https://ocw.mit.edu*.

Pearce, R. H. and Forbes, M. (2018). Disaggregated Benders decomposition and branch-and-cut for solving the budget-constrained dynamic uncapacitated facility location and network design problem. *European Journal of Operational Research*, 270(1):78–88.

Pearce, R. H. and Hons, B. S. (2019). Towards a general formulation of lazy constraints.

Pinedo, M. L. (2016). *Scheduling Theory, Algorithms, and Systems*. Springer, 5 edition.

Quelhas, A., Gil, E., McCalley, J. D., and Ryan, S. M. (2007). A multiperiod generalized network flow model of the U.S. integrated energy system: Part I - Model description. *IEEE Transactions on Power Systems*, 22(2):829–836.

Rahmaniani, R., Crainic, T. G., Gendreau, M., and Rei, W. (2017). The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817.

Ruth, M. F. and Kroposki, B. (2014). Energy Systems Integration : An Evolving Energy Paradigm. *The Electricity Journal*, (January 2018).

Samsatli, S. and Samsatli, N. J. (2018). A general mixed integer linear programming model for the design and operation of integrated urban energy systems. *Journal of Cleaner Production*, 191:458–479.

van Beuzekom, I., Gibescu, M., Pinson, P., and Slootweg, J. G. (2017). Optimal planning of integrated multi-energy systems. *2017 IEEE Manchester PowerTech, Powertech 2017*, 1.

Van Beuzekom, I., Mazairac, L. A. J., Gibescu, M., and Slootweg, J. G. (2016). Optimal Design and Operation of an Integrated Multi- Energy System for Smart Cities. *2016 IEEE International Energy Conference (ENERGYCON)*.

Van Beuzekom, I., Member, S., Nijhuis, M., Member, S., Hodge, B., Member, S., Pinson, P., Member, S., and Slootweg, J. G. (2020). Optimal Planning of Integrated Electricity , Gas and Heat Systems : a Multigrid Approach.

Wakui, T., Kinoshita, T., and Yokoyama, R. (2014). A mixed-integer linear programming approach for cogeneration-based residential energy supply networks with power and heat interchanges. *Energy*, 68:29–46.

Weber, C. and Shah, N. (2011). Optimisation based design of a district energy system for an eco-town in the United Kingdom. *Energy*, 36(2):1292–1308.

Winston, W. L. and Goldber, J. B. (2004). *Operations research*. Brooks/Cole, 4 edition.

Wouters, C., Fraga, E. S., and James, A. M. (2015). An energy integrated, multi-microgrid, MILP (mixed-integer linear programming) approach for residential distributed energy system planning - A South Australian case-study. *Energy*, 85:30–44.

A

Additional figures

This chapter provides visualizations of all of the experiments as conducted in Chapter 7.

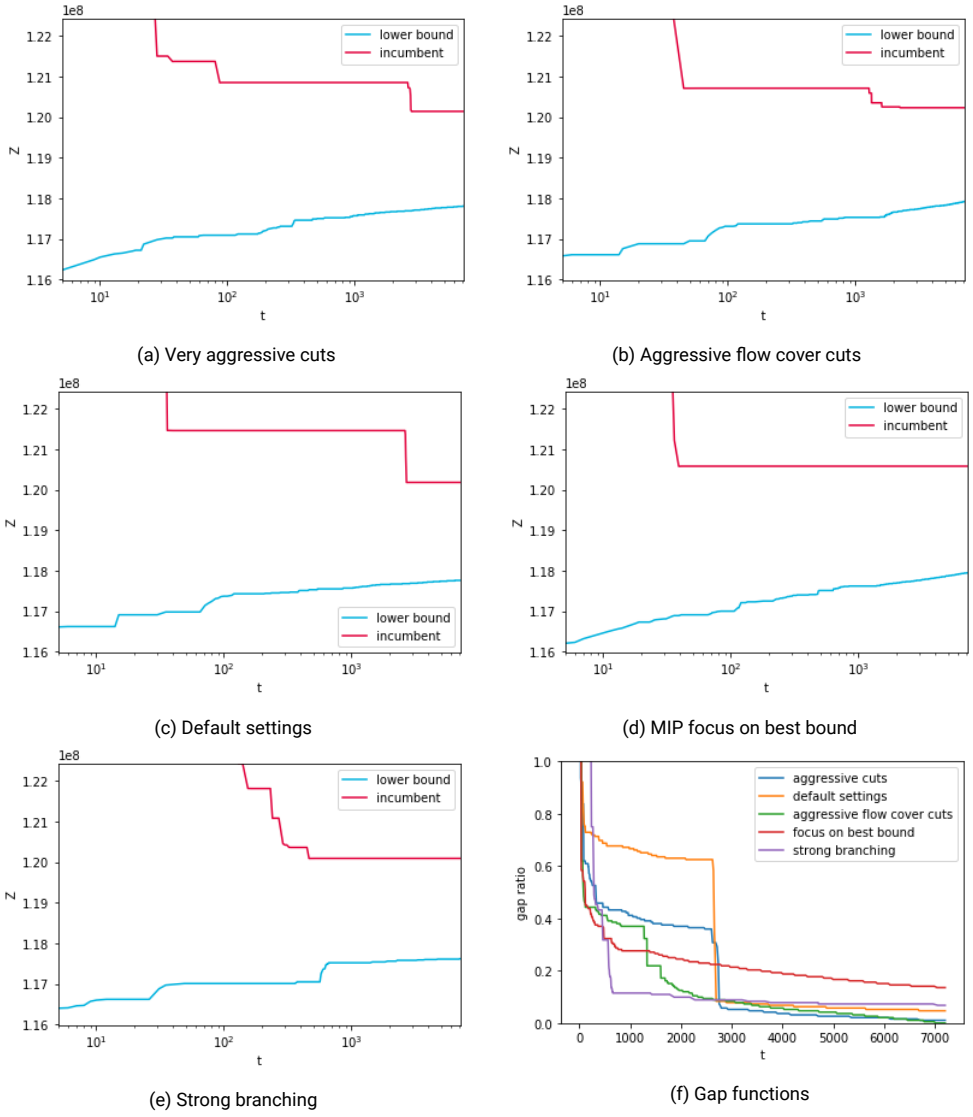
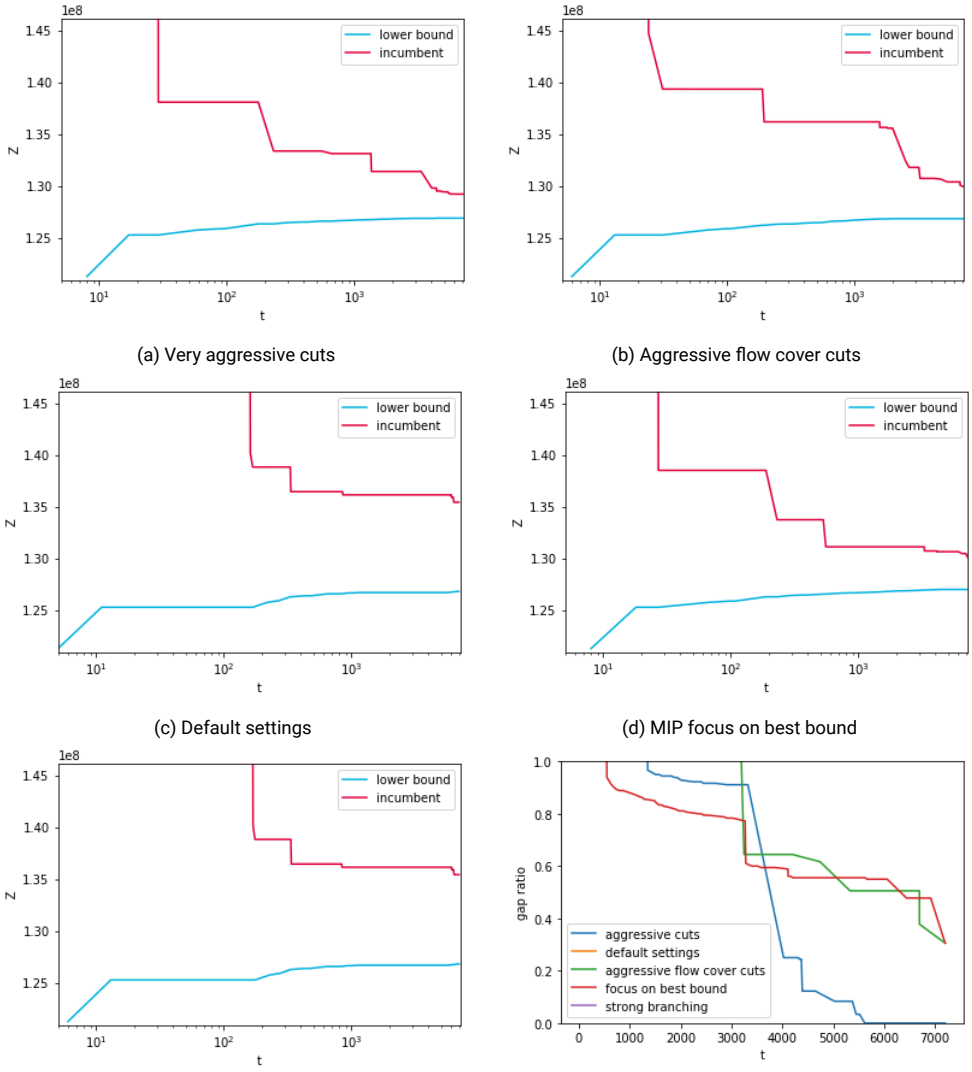


Figure A.1: Graphs corresponding to the evolution of the bounds for the instance of reg7 and their corresponding gap functions



(a) Very aggressive cuts

(b) Aggressive flow cover cuts

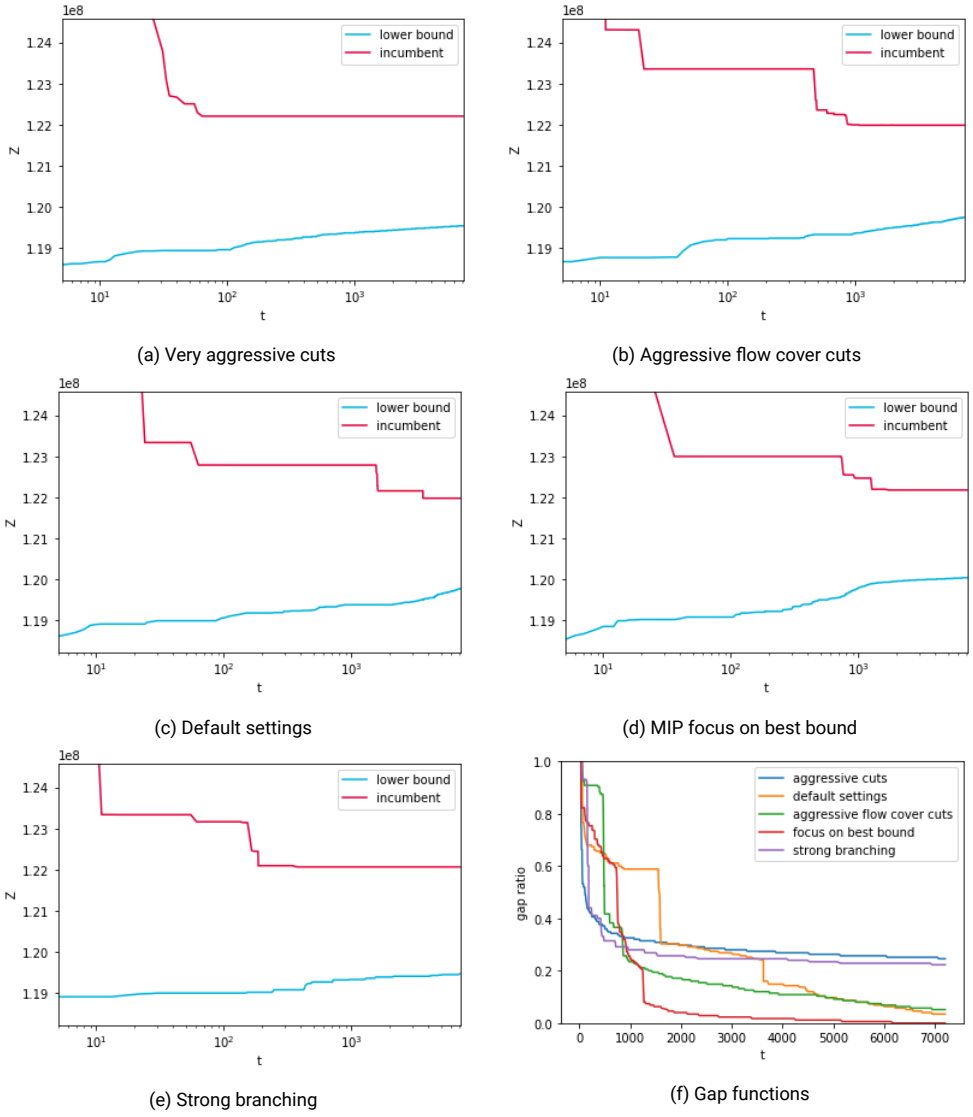
(c) Default settings

(d) MIP focus on best bound

(e) Strong branching

(f) Gap functions

Figure A.2: Graphs corresponding to the evolution of the bounds for the instance of reg28 and their corresponding gap functions



(a) Very aggressive cuts

(b) Aggressive flow cover cuts

(c) Default settings

(d) MIP focus on best bound

(e) Strong branching

(f) Gap functions

Figure A.3: Graphs corresponding to the evolution of the bounds for the instance of ud7 and their corresponding gap functions

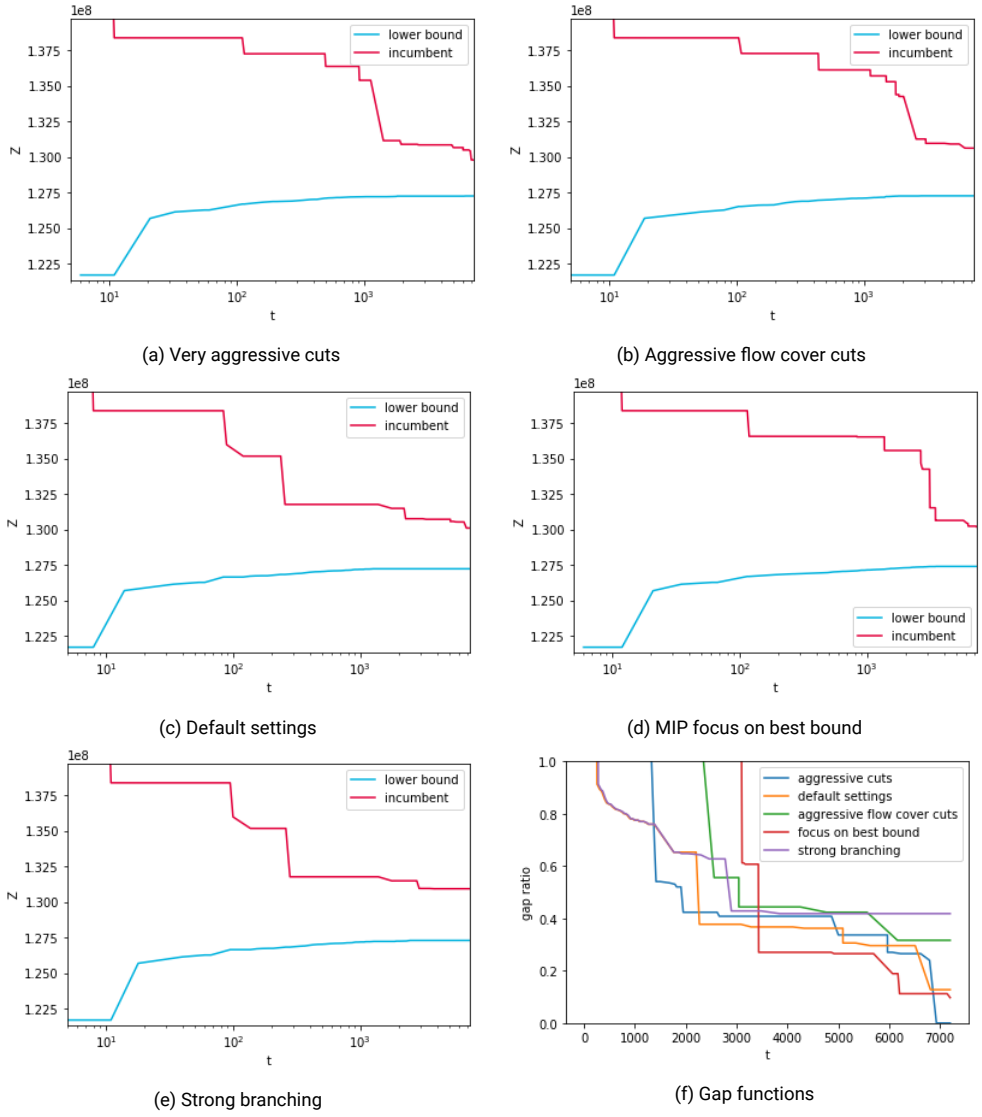
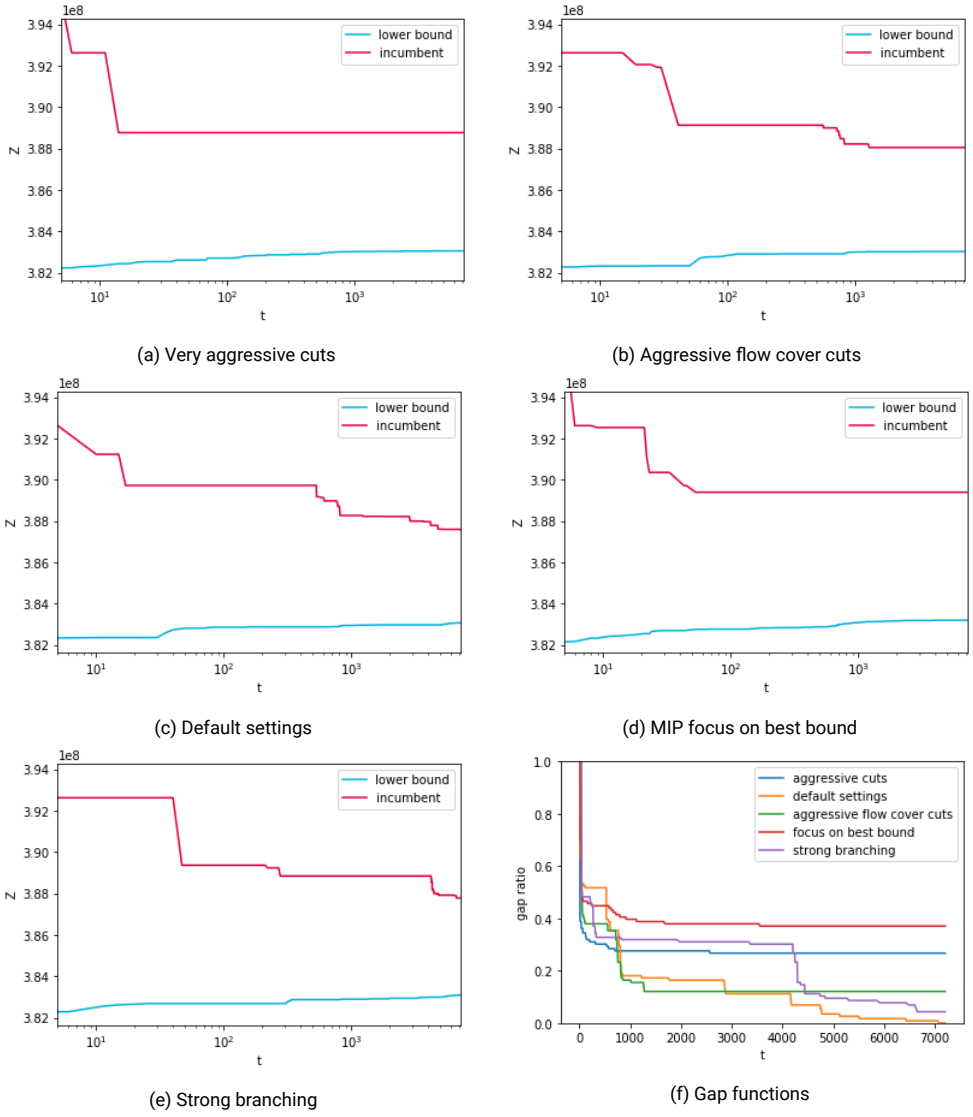


Figure A.4: Graphs corresponding to the evolution of the bounds for the instance of ud28 and their corresponding gap functions



(a) Very aggressive cuts

(b) Aggressive flow cover cuts

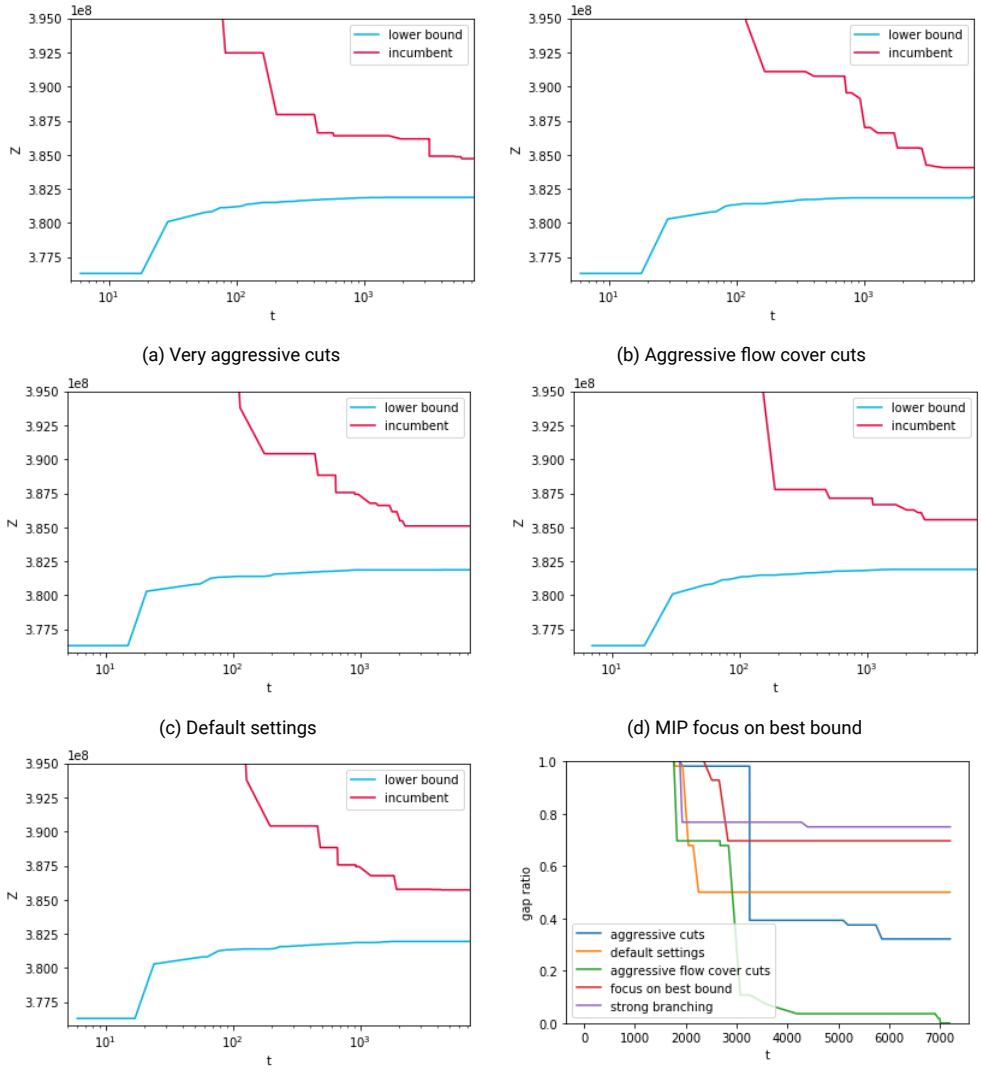
(c) Default settings

(d) MIP focus on best bound

(e) Strong branching

(f) Gap functions

Figure A.5: Graphs corresponding to the evolution of the bounds for the instance of gas7 and their corresponding gap functions



(a) Very aggressive cuts

(b) Aggressive flow cover cuts

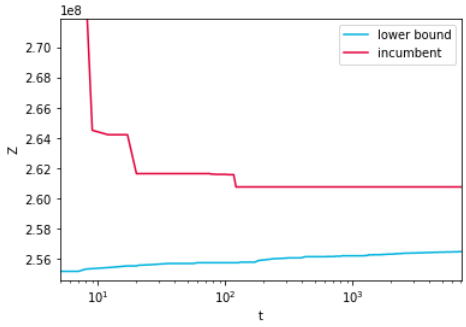
(c) Default settings

(d) MIP focus on best bound

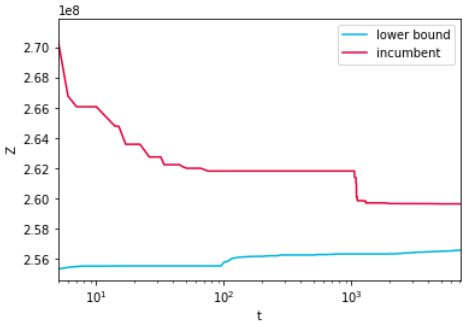
(e) Strong branching

(f) Gap functions

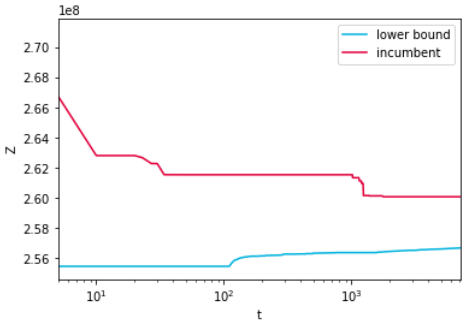
Figure A.6: Graphs corresponding to the evolution of the bounds for the instance of gas28 and their corresponding gap functions



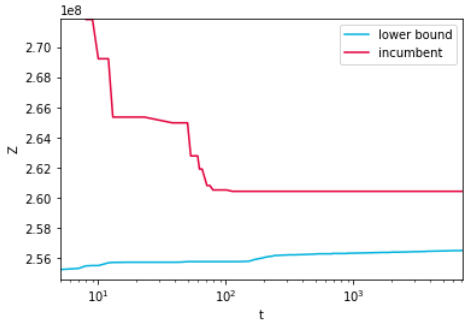
(a) Very aggressive cuts



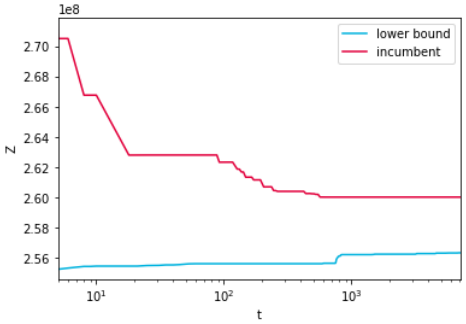
(b) Aggressive flow cover cuts



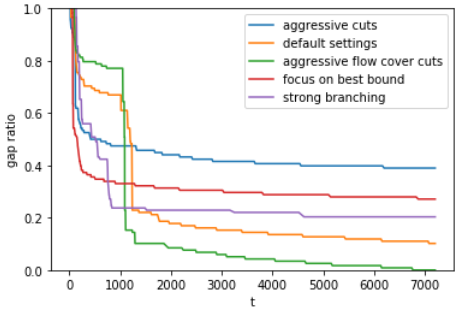
(c) Default settings



(d) MIP focus on best bound

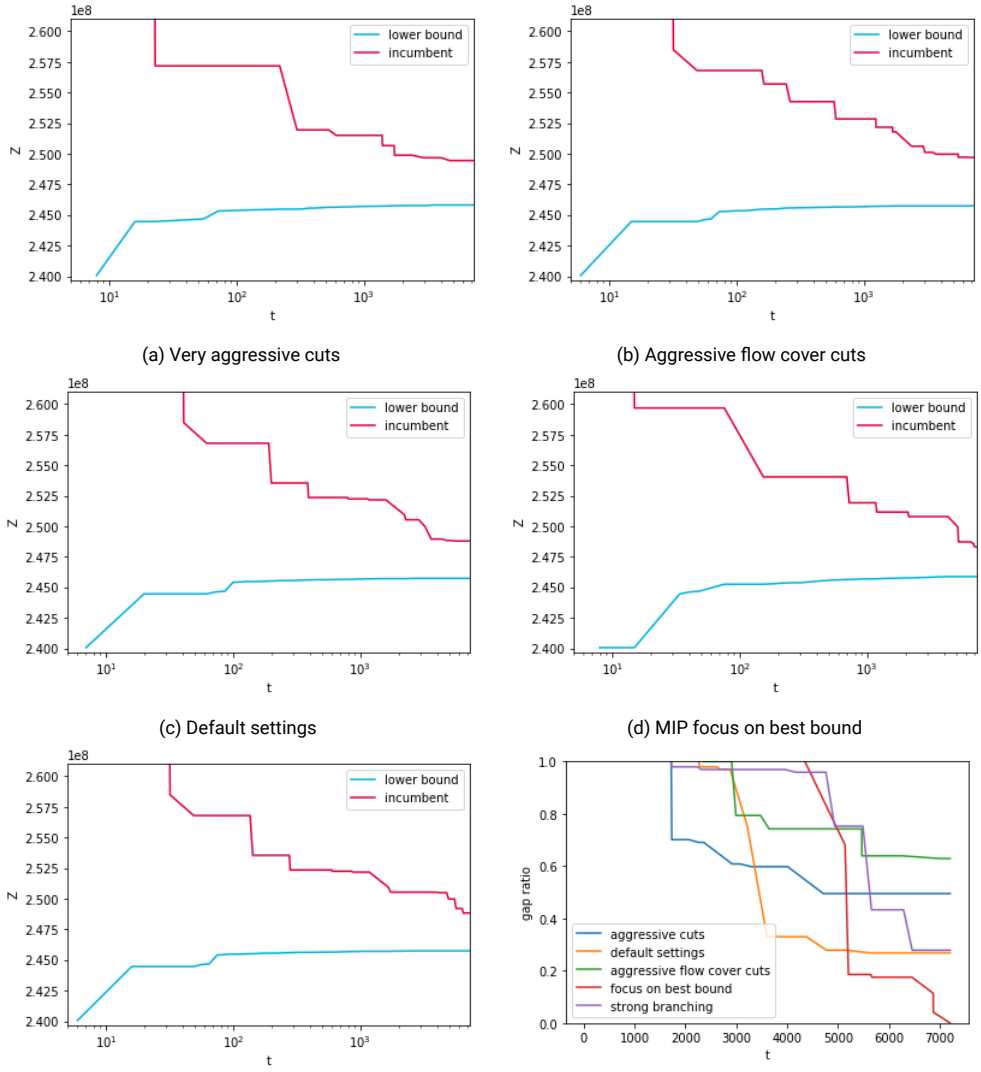


(e) Strong branching



(f) Gap functions

Figure A.7: Graphs corresponding to the evolution of the bounds for the instance of dem17 and their corresponding gap functions



(a) Very aggressive cuts

(b) Aggressive flow cover cuts

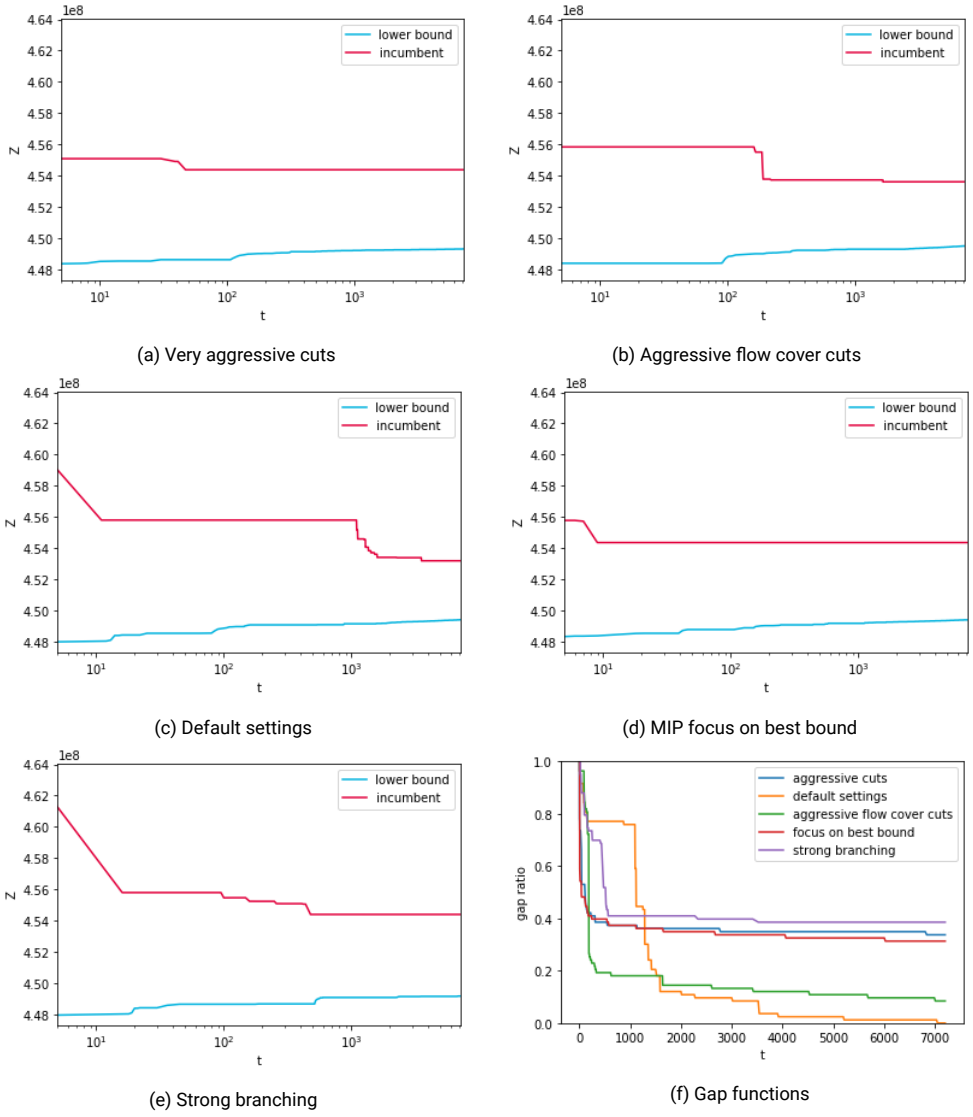
(c) Default settings

(d) MIP focus on best bound

(e) Strong branching

(f) Gap functions

Figure A.8: Graphs corresponding to the evolution of the bounds for the instance of dem128 and their corresponding gap functions



(a) Very aggressive cuts

(b) Aggressive flow cover cuts

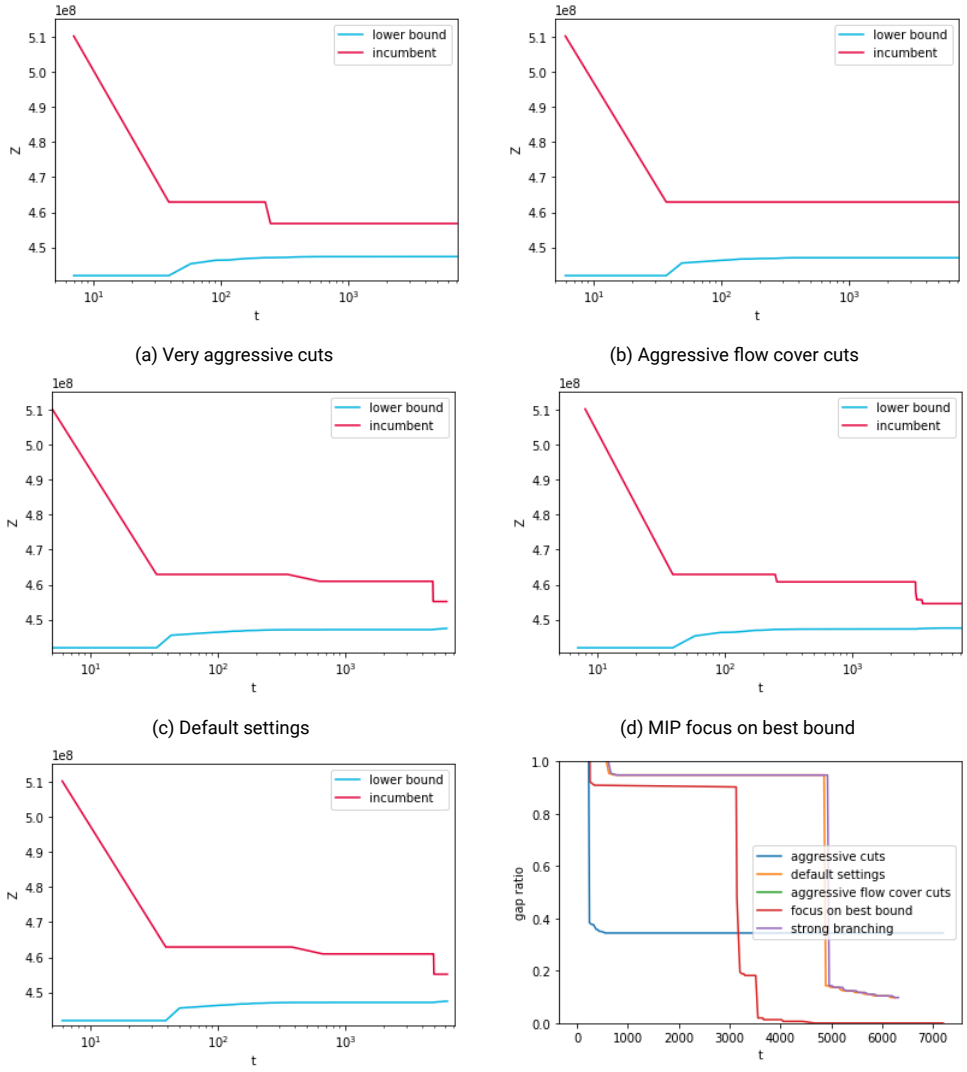
(c) Default settings

(d) MIP focus on best bound

(e) Strong branching

(f) Gap functions

Figure A.9: Graphs corresponding to the evolution of the bounds for the instance of dem27 and their corresponding gap functions



(a) Very aggressive cuts

(b) Aggressive flow cover cuts

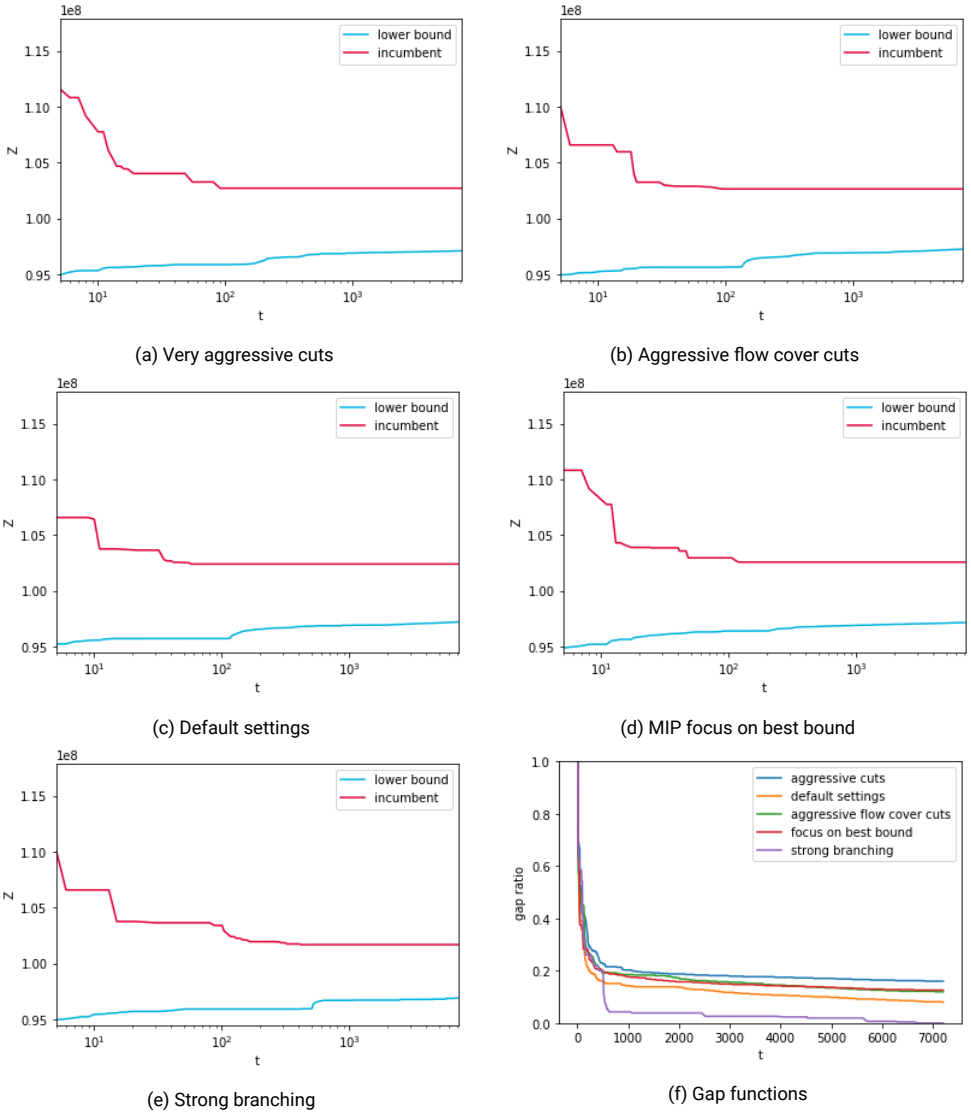
(c) Default settings

(d) MIP focus on best bound

(e) Strong branching

(f) Gap functions

Figure A.10: Graphs corresponding to the evolution of the bounds for the instance of dem228 and their corresponding gap functions



(a) Very aggressive cuts

(b) Aggressive flow cover cuts

(c) Default settings

(d) MIP focus on best bound

(e) Strong branching

(f) Gap functions

Figure A.11: Graphs corresponding to the evolution of the bounds for the instance of cc7 and their corresponding gap functions

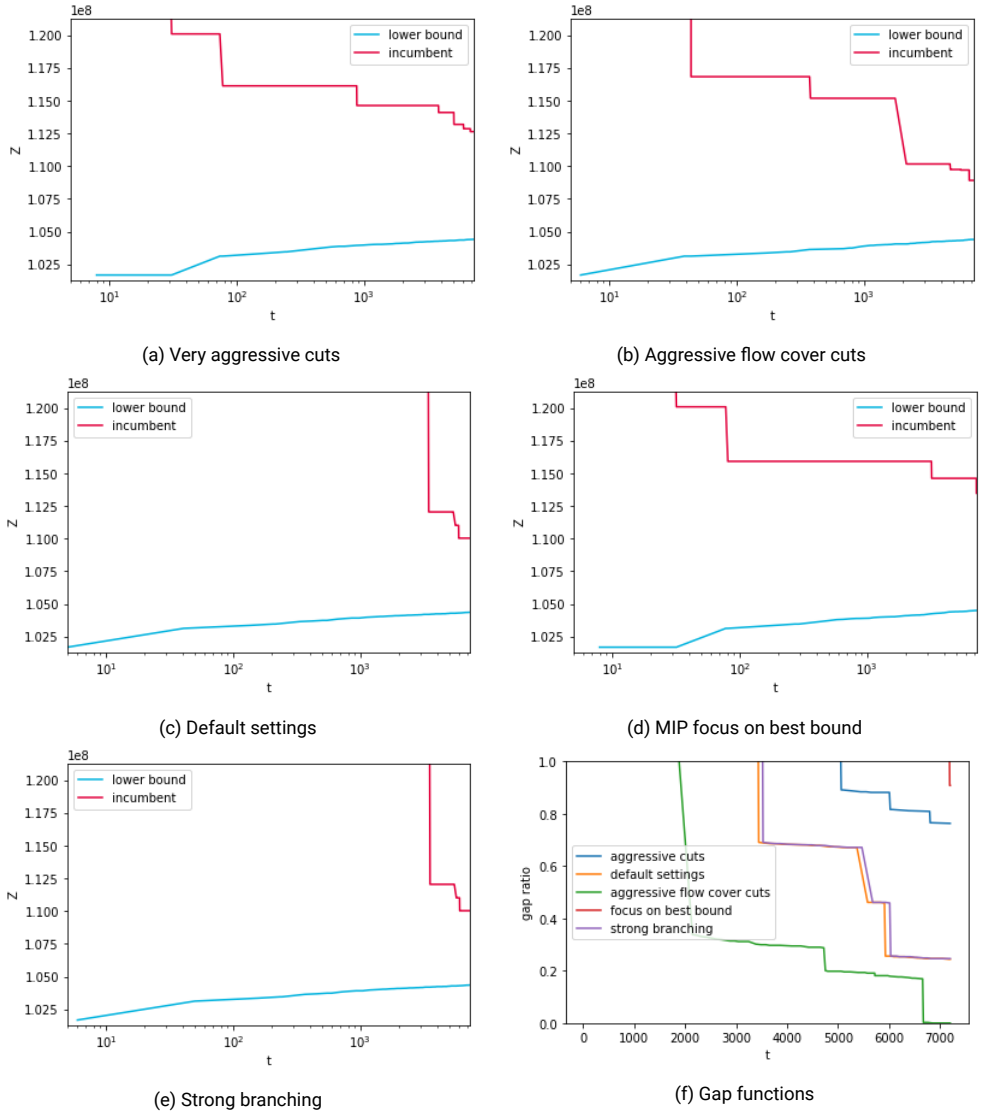
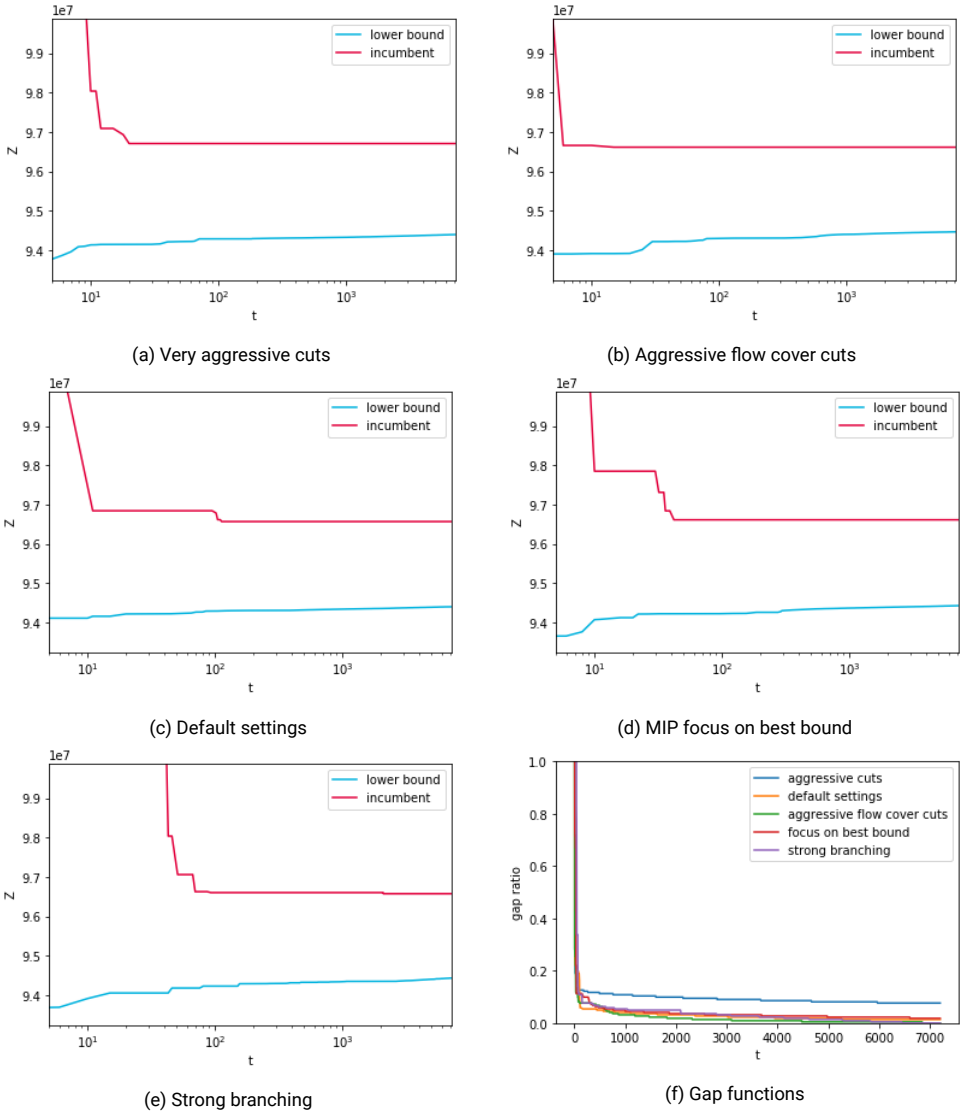


Figure A.12: Graphs corresponding to the evolution of the bounds for the instance of cc28 and their corresponding gap functions



(a) Very aggressive cuts

(b) Aggressive flow cover cuts

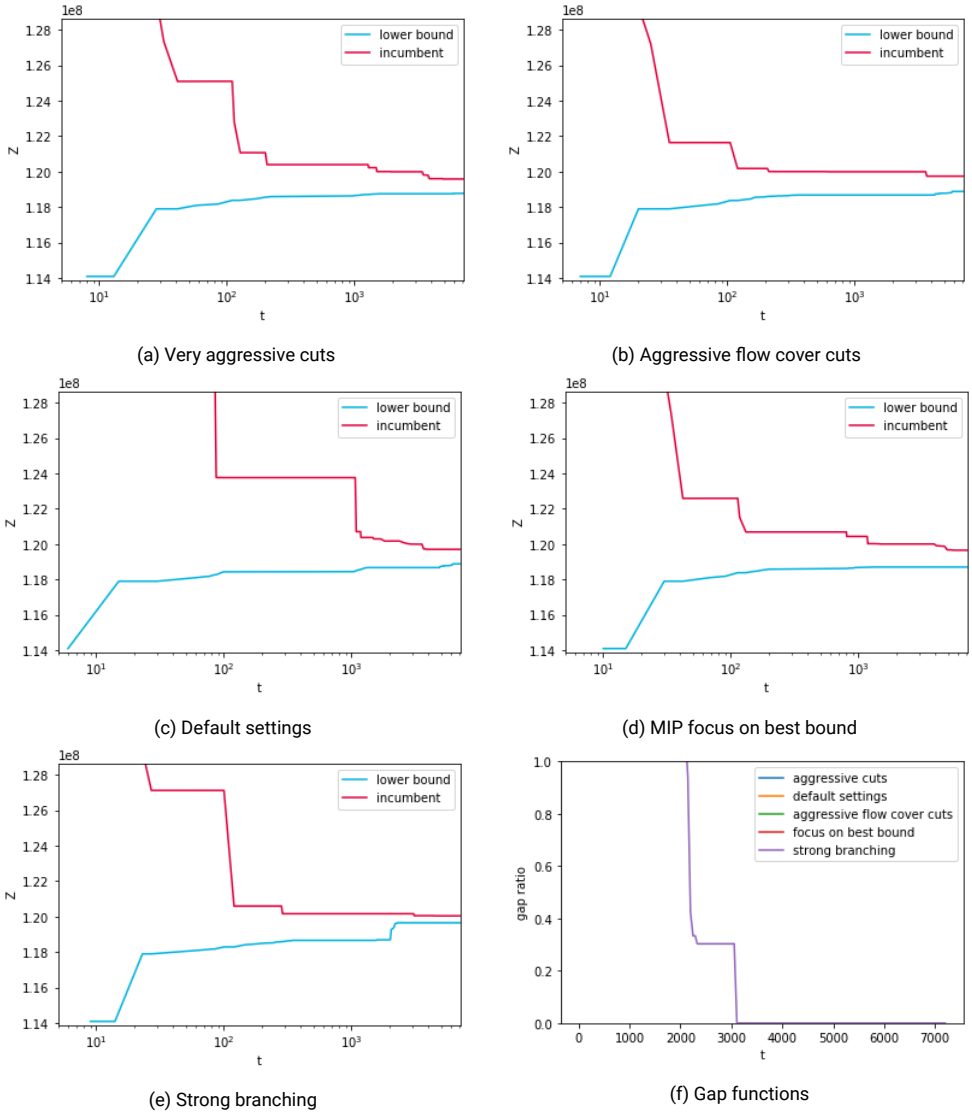
(c) Default settings

(d) MIP focus on best bound

(e) Strong branching

(f) Gap functions

Figure A.13: Graphs corresponding to the evolution of the bounds for the instance of bf7 and their corresponding gap functions



(a) Very aggressive cuts

(b) Aggressive flow cover cuts

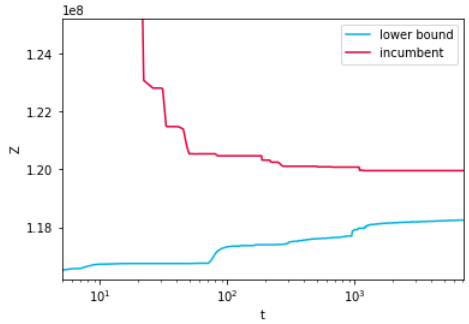
(c) Default settings

(d) MIP focus on best bound

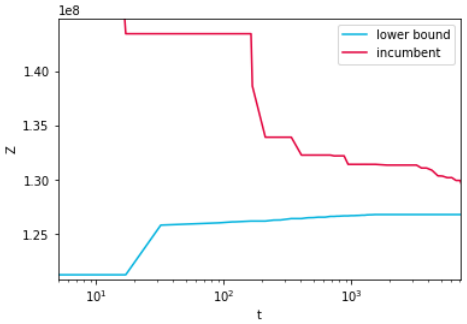
(e) Strong branching

(f) Gap functions

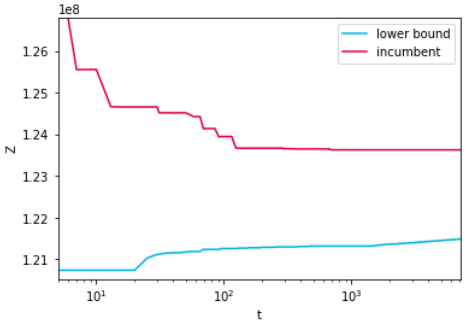
Figure A.14: Graphs corresponding to the evolution of the bounds for the instance of bf28 and their corresponding gap functions



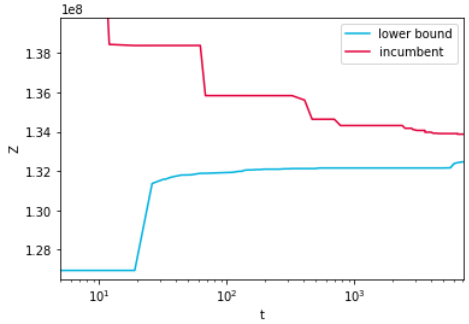
(a) reg7



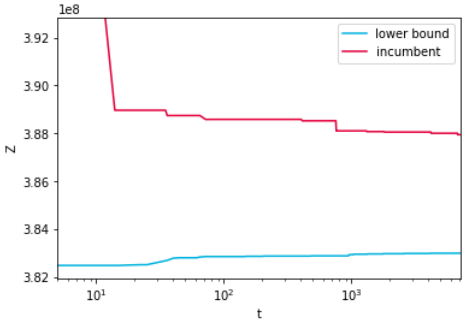
(b) reg28



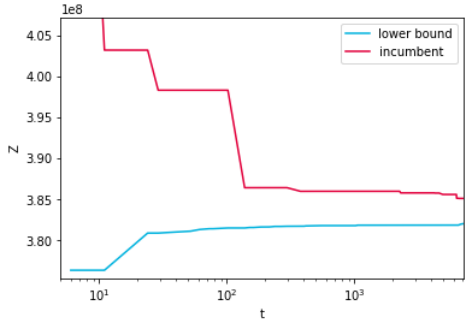
(c) ud7



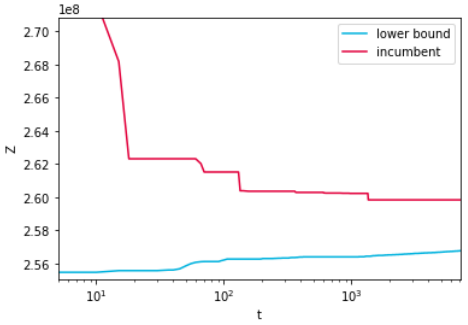
(d) ud28



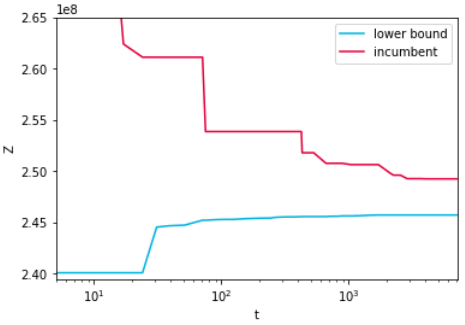
(e) gas7



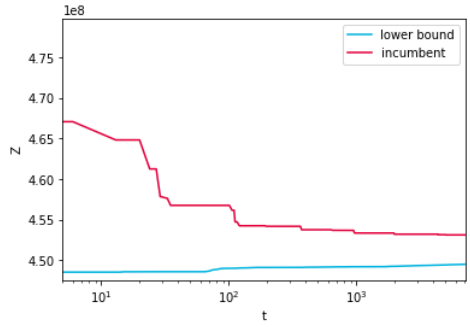
(f) gas28



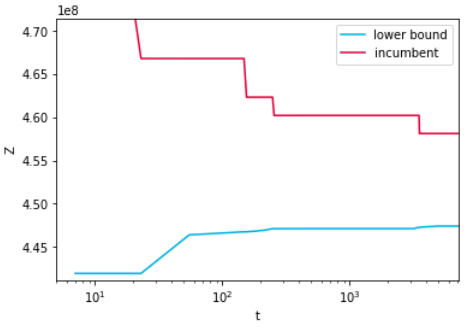
(g) dem17



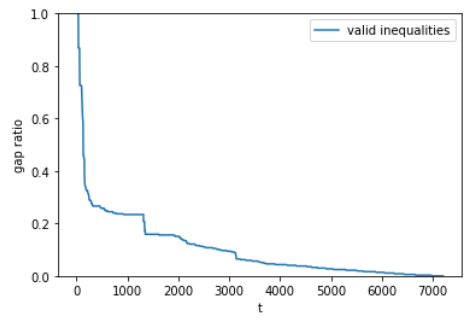
(h) dem128



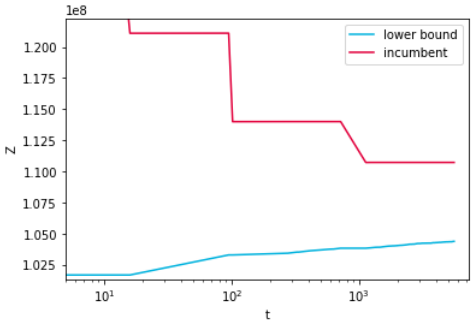
(i) dem27



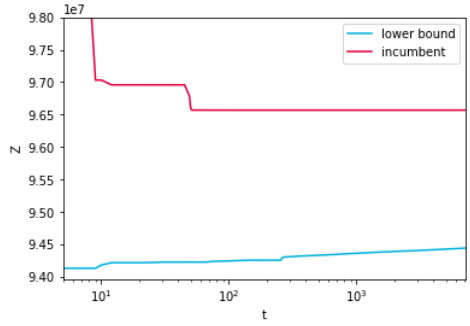
(j) dem228



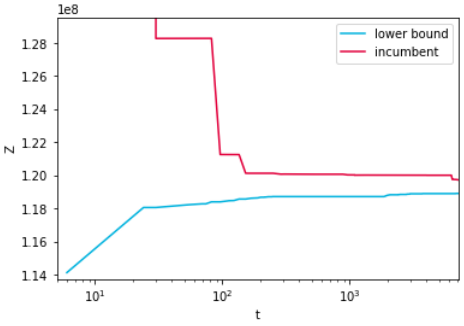
(k) cc7



(l) cc28

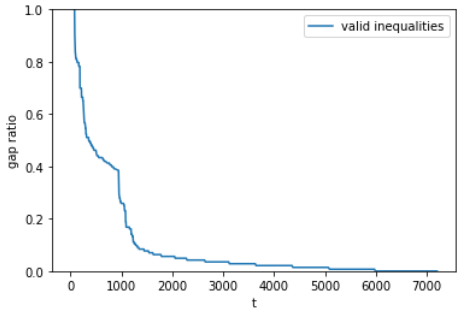


(m) bf7

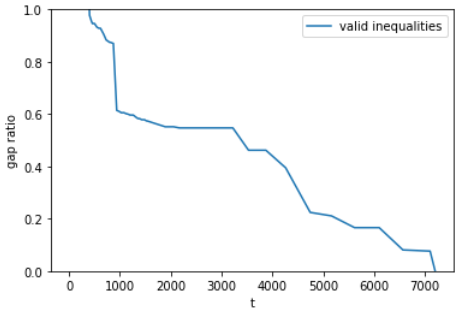


(n) bf28

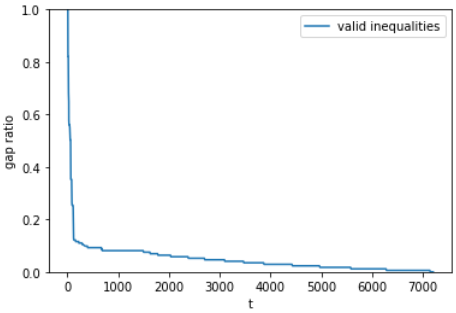
Figure A.15: Graphs corresponding to the evolution of the bounds for the set of instances when the valid inequalities are applied (and the solvers settings are set to default)



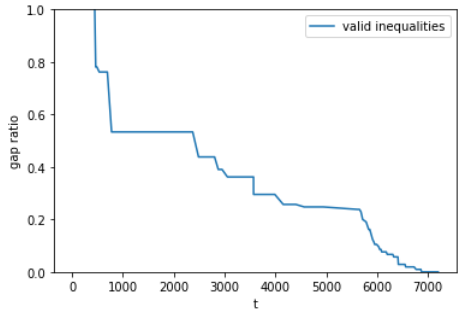
(a) reg7



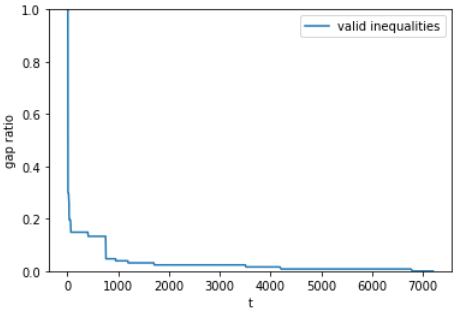
(b) reg28



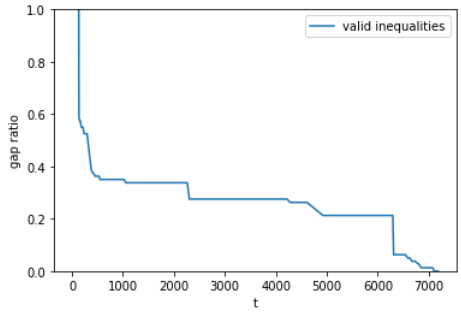
(c) ud7



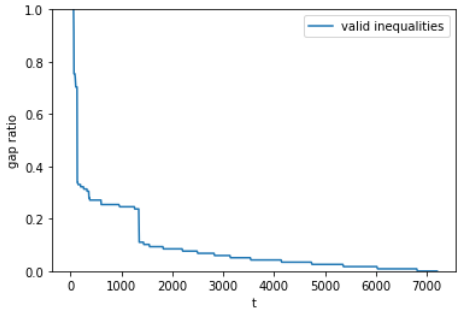
(d) ud28



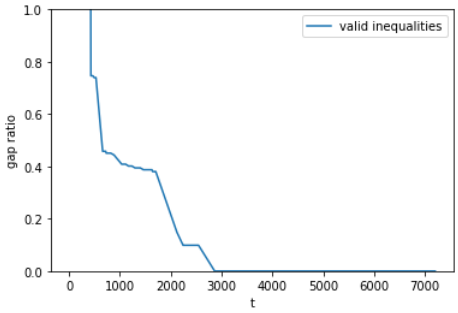
(e) gas7



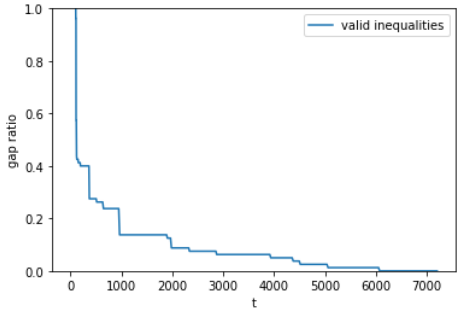
(f) gas28



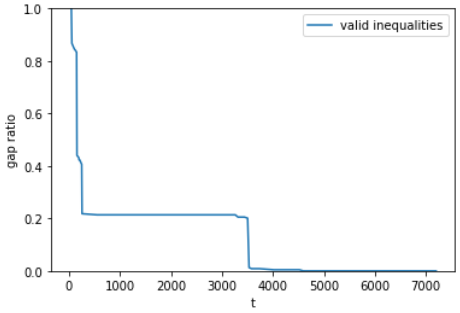
(g) dem17



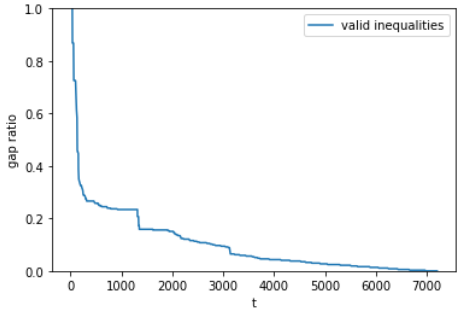
(h) dem128



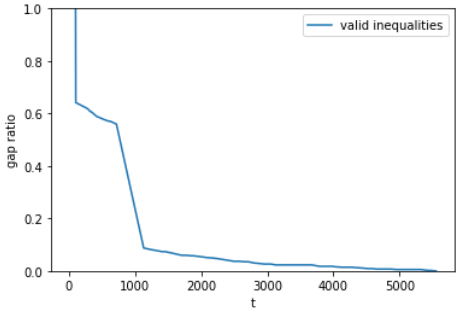
(i) dem27



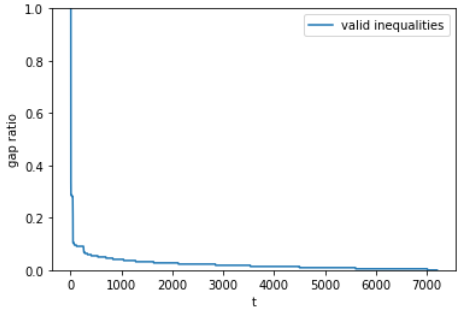
(j) dem228



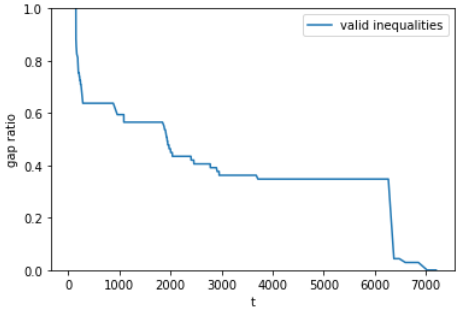
(k) cc7



(l) cc28

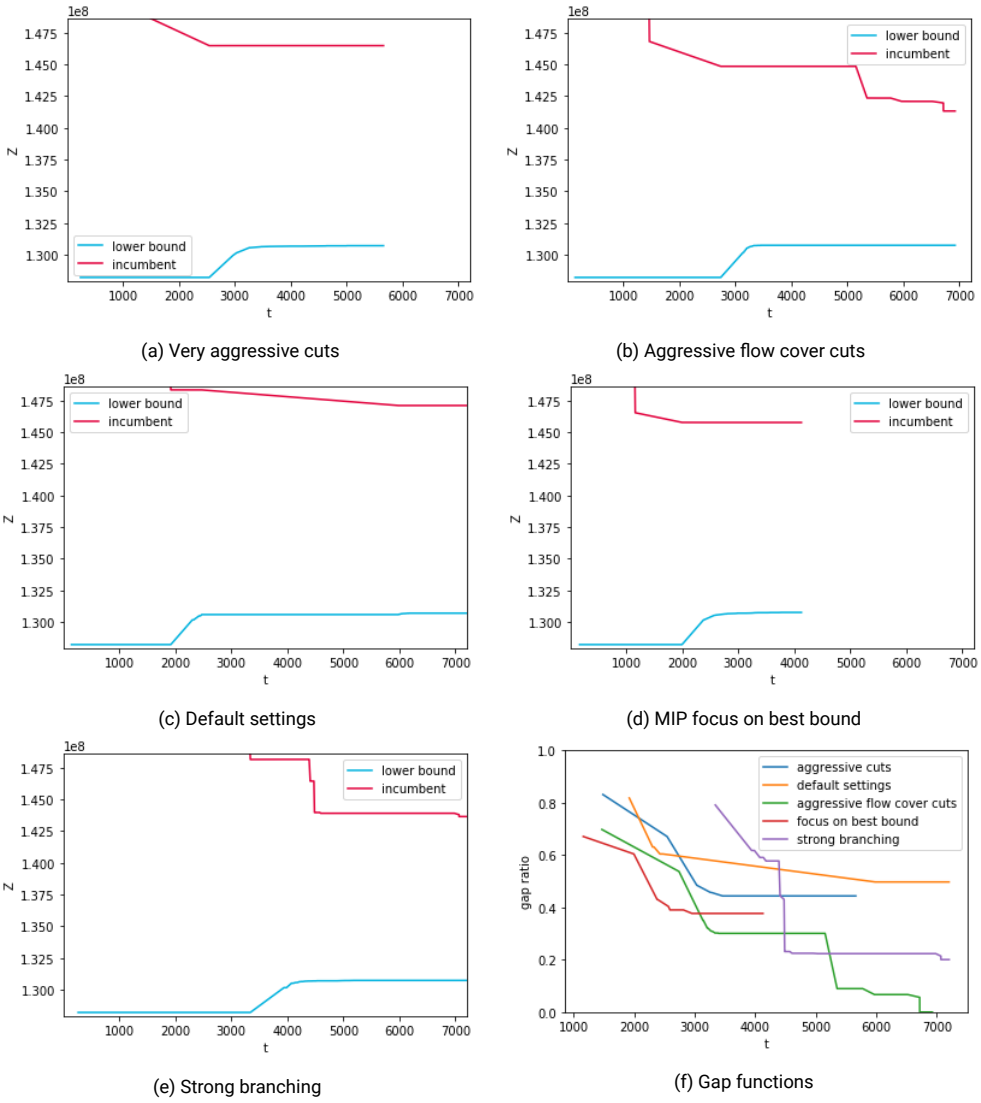


(m) bf7



(n) bf28

Figure A.16: Graphs corresponding to the evolution of the gap functions for the set of instances when the valid inequalities are applied (and the solvers settings are set to default)



(a) Very aggressive cuts

(b) Aggressive flow cover cuts

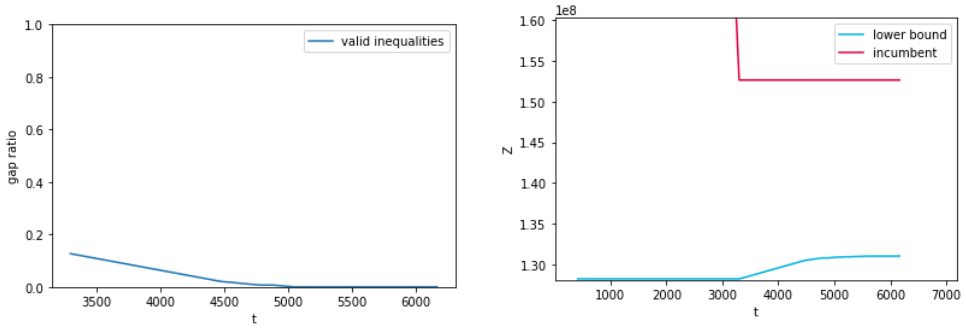
(c) Default settings

(d) MIP focus on best bound

(e) Strong branching

(f) Gap functions

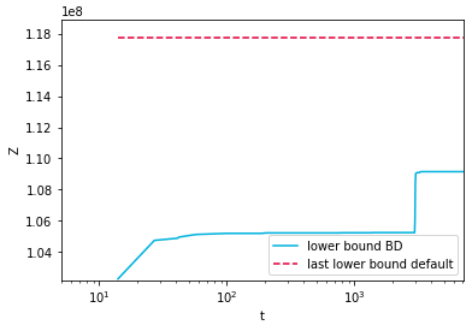
Figure A.17: Graphs corresponding to the evolution of the bounds for the instance of reg110 and their corresponding gap functions



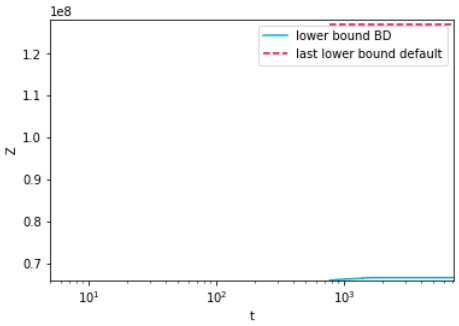
(a) gap ratio of instance reg110 with valid inequalities

(b) bound evolutions of instance reg110 with valid inequalities

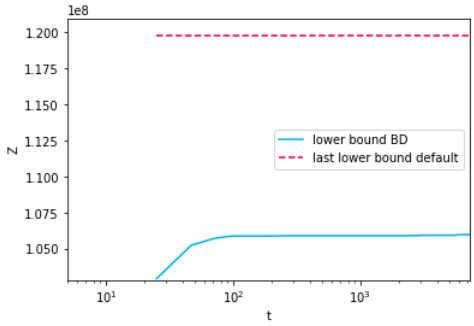
??



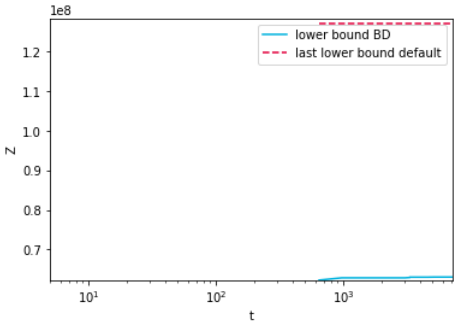
(a) reg7



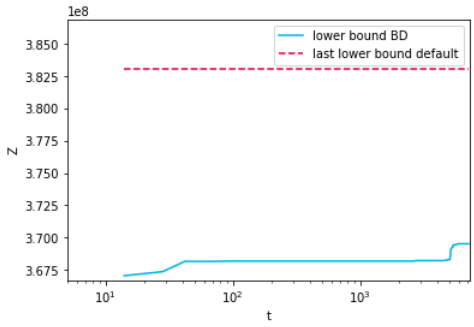
(b) reg28



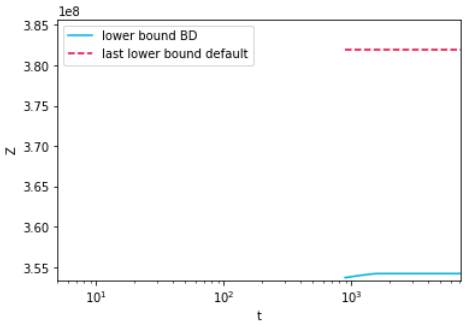
(c) ud7



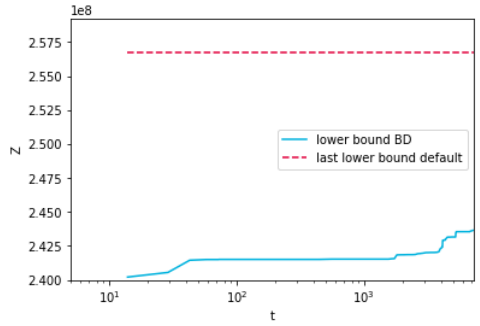
(d) ud28



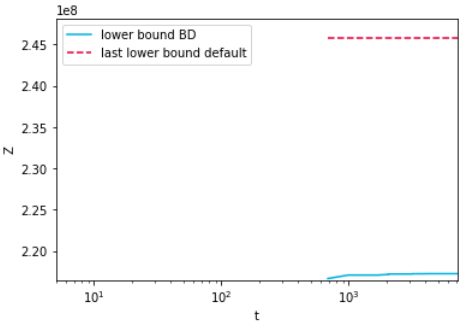
(e) gas7



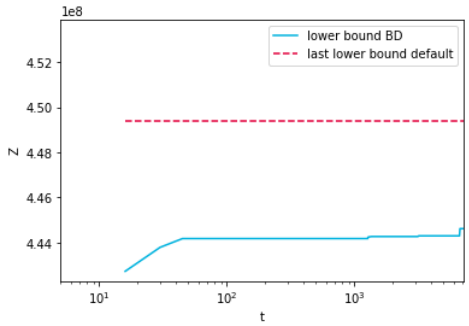
(f) gas28



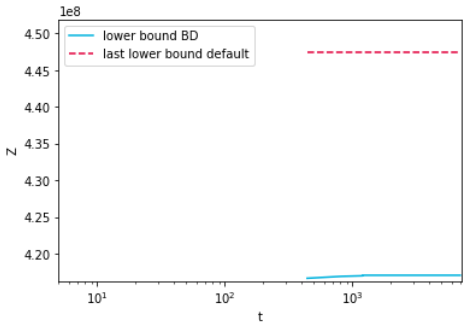
(g) dem17



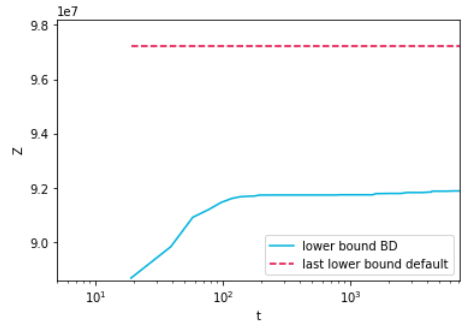
(h) dem128



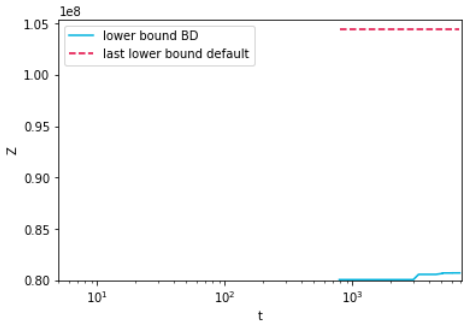
(i) dem27



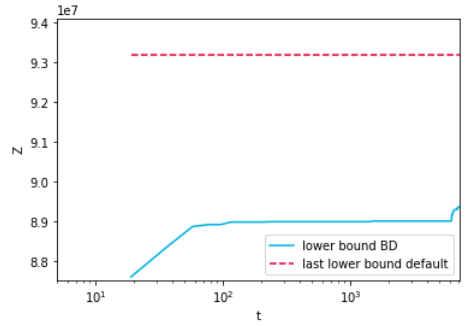
(j) dem228



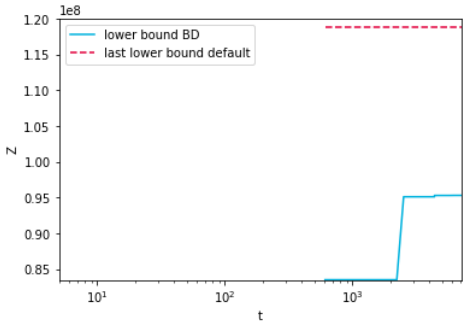
(k) cc7



(l) cc28

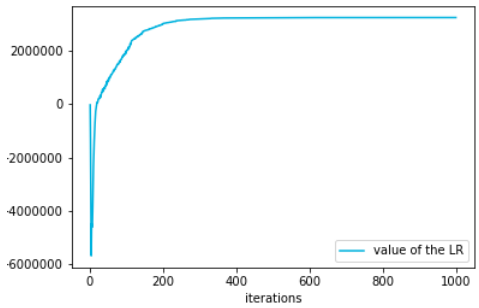


(m) bf7

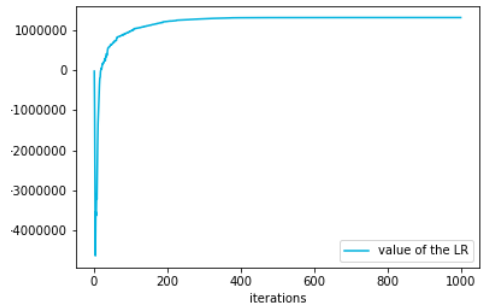


(n) bf28

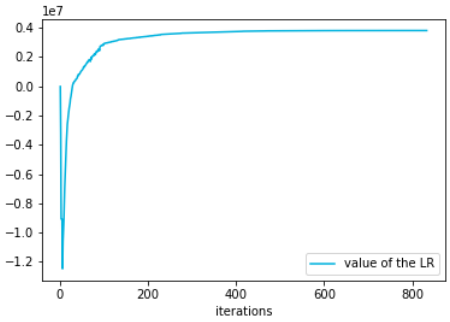
Figure A.19: Graphs corresponding to the evolution of lower bounds in the BD algorithm, plotted against the lower bound that was found in the last iteration of the regular branch and bound algorithm for the original problem



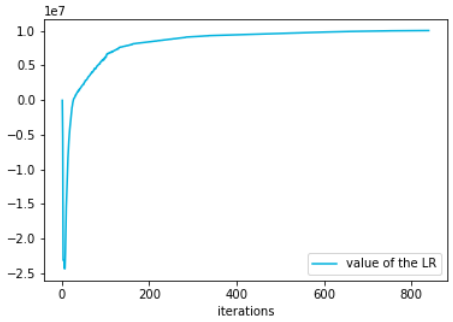
(a) reg7



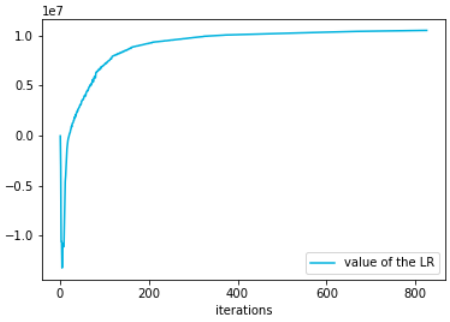
(b) cc7



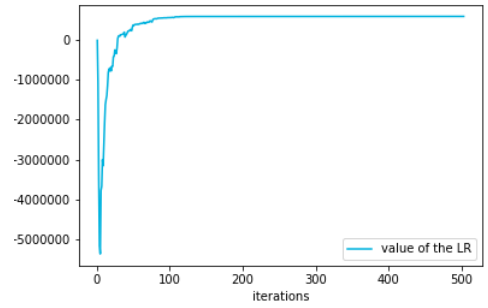
(c) dem17



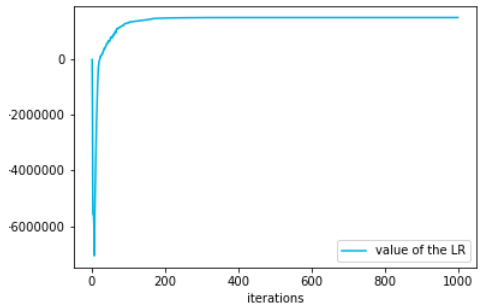
(d) dem27



(e) gas7



(f) bf7



(g) ud7

Figure A.20: Evolution of the Lagrangean Relaxation function for the 7 node instances

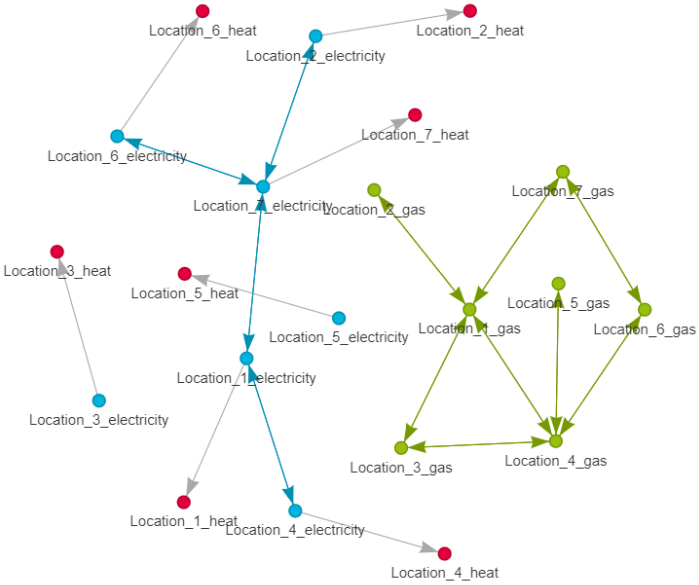


Figure A.21: A network generated for a 7 node instance

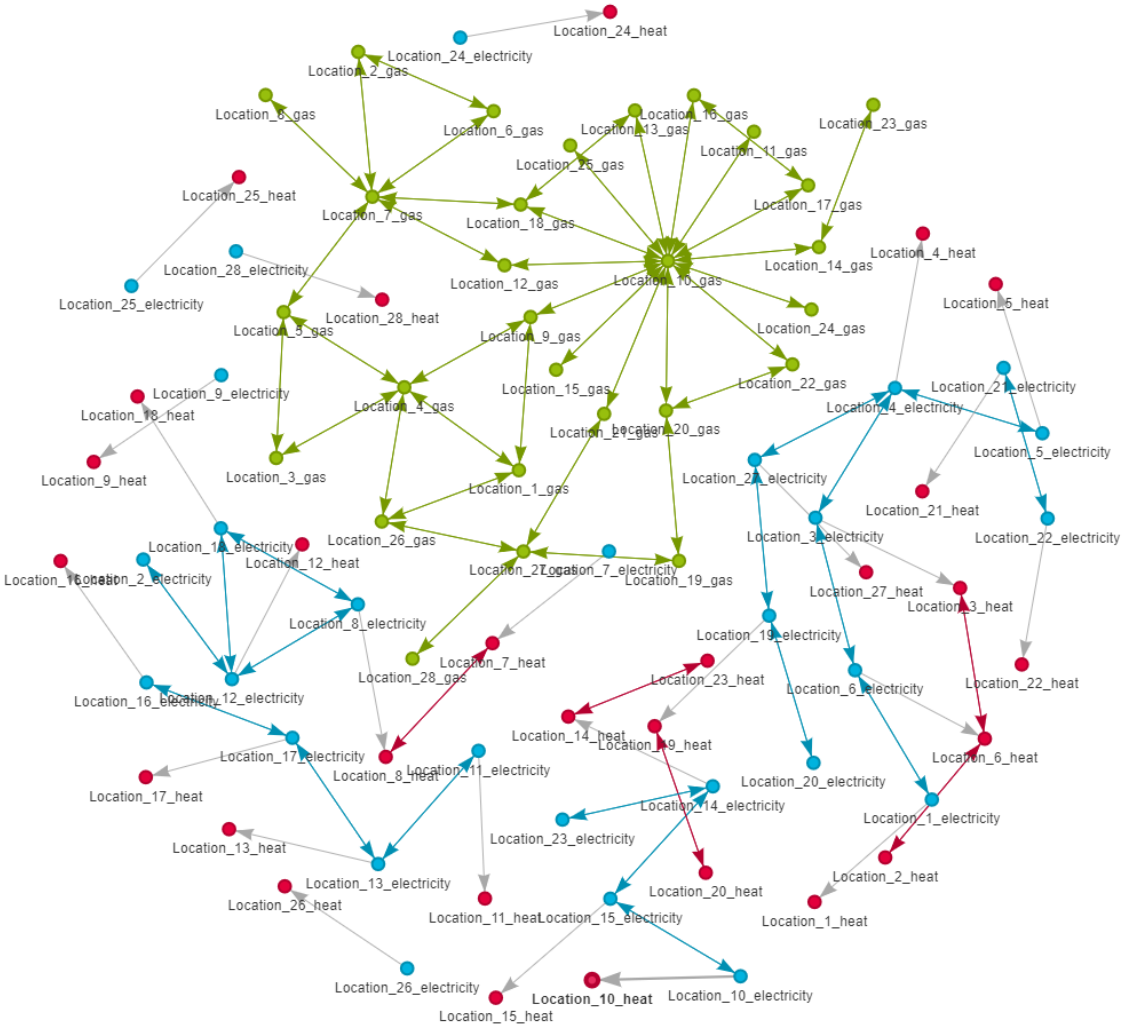


Figure A.22: A network generated for a 28 node instance

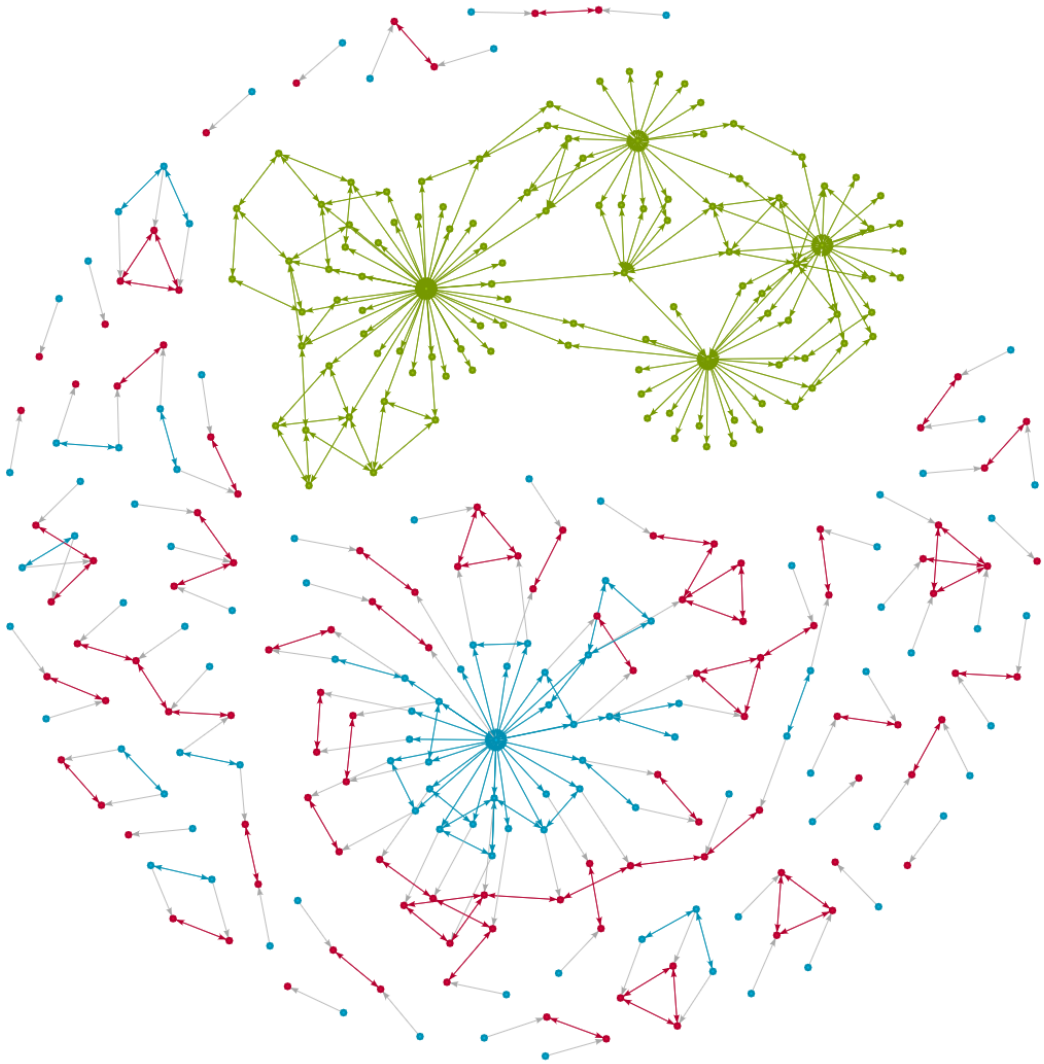


Figure A.23: A network generated for a 110 node instance